Determining partition elements with feedback constraints

Breckon, Thomas Joseph

Monterey, California. Naval Postgraduate School

DETERMINING PARTITION ELEMENTS WITH FEEDBACK CONTRAINTS

Thomas Joseph Breckon

United States
Naval Postgraduate School

THESIS

DETERMINING PARTITION ELEMENTS

WITH FEEDBACK CONSTRAINTS

by

Thomas Joseph Breckon

June 1970

Determining Partition Elements with Feedback Constraints

by

Thomas Joseph Breckon
Lieutenant (junior grade), United States Navy
B.A., University of California at Los Angeles, 1969

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1970

ABSTRACT

The partition problem is that step in the layout problem in which it must be decided which of the elementary digital circuits are to be coalesced into a single, electronic package. A solution of the partition problem must satisfy constraints on the maximum number of elementary circuits that can be put into a single package, and on the number of external connections that can be attached to the package.

A solution due to Lawler et al [Ref. 12], which minimizes delay caused by clustering electronic elements, is extended to cyclic networks. A new algorithm to extract from a graph the maximal strongly-connected subgraphs (lobes) is developed, and a new approach to clustering the digital elements of a lobe is presented. The digital circuit is represented by a bipartite graph, and solutions are expressed in terms of graph theory.

TABLE OF CONTENTS

I. INTRODUCTION

A. THE DESIGN PROCESS

The design of a digital electronic circuit takes place in several stages. In the first stage the designer determines precisely the relations between the inputs to the digital circuit and the desired outputs. Based on this analysis, and dependent also on certain physical characteristics of the electronic medium that is used, the fundamental logic set of the digital circuit is defined. In the fundamental logic set are groups of elementary circuit elements such as "or," "and" or "nor" gates, combined in small electronic packages such as miniturized integrated circuits or semiconductor chips (substrates). Members of the fundamental logic set are the most basic replaceable logic elements of the digital circuit, and it is from these that the digital circuit is built.

Based also on the analysis of inputs and desired outputs is the determination of the specifications of the digital circuit. After the specifications are complete, the synthesis of the digital circuit takes place. Following the specifications, elements of the fundamental logic set are interconnected to form a representation of a digital circuit. One of the prime objectives of the logic designer is simplicity.

The logic design is produced relatively independently of the physical structure of the digital circuit. The engineer takes as his input the results of the logic designer, and

produces the digital circuit according to the physical and electronic characteristics of the circuit elements and their interconnections.

Kodres [Ref. 10] refers to that step of the logic design process, in which it must be determined how elementary logic elements are to be interconnected, as the layout problem. The layout problem is concerned with such questions as how many electronic elements can be placed in a single electronic package, in what relative locations they should be placed within the package, and in what manner can several packages be interconnected. The layout problem is dependent on physical properties such as the interconnection of electronic components, the number of electronic components or the number of input or output signals at a functional component.

The layout problem can be applied to the design of digital circuits at any of several levels of increasing scope. For example, the results of the theory developed in general terms can be applied with little modification to the design of members of the fundamental logic set and the layout of gates within such a small package. Application can also be made to the clustering of fundamental logic set elements within larger electronic modules.

B. THE PARTITION PROBLEM

The partition problem is that step in the layout problem in which it must be decided which of the elementary circuits are to be combined into a single electronic package. The

solution of the partition problem must satisfy a number of constraints. These constraints are divided into two categories: packaging constraints and performance constraints.

Most important to the designer of packages is to keep within space limitations. Thus the number of elementary circuits in a single package is limited by their very size. Closely associated with the problem of limited space is a limit on the number of external connections that are possible for a package of a certain size.

The limit on the number of external connections is more or less critical depending on the application of the partition problem. In the case of a semiconductor chip, the electronic elements are placed on the chip chemically. These elements are extremely small. In comparison, the external connections to the chip are physically attached to the package that contains the chip by means of solder connections, and are thus an order of magnitude larger than the internal logic elements. In this case, the external connection constraint is very critical.

If the electronic package is the size of a replaceable module, or circuit card, in a computer, then the external connection constraint is less critical. In this case the external connections are in the form of small pins that line one edge of the card, and that are plugged into the computer's superstructure.

Packaging constraints are due to physical and economic considerations. There is, however, an additional constraint

that is due to performance characteristics. It is desirable to reduce the amount of electronic delay through the network. Delay is defined in terms of a function proportional to the total wire length, or the wire length in the longest closed path, or the wire length from an input of a network to an output.

The problem at hand, which is the variant of the partition problem that Lawler et al [Ref. 12] considered, is a minimization problem. Given a digital circuit specified by the logic design, assume that a maximum of M elementary functional circuits can be accommodated in an electronic package, and that a maximum of P external connections can be accommodated. Assume also that in a resultant network of completed electronic packages, no delay is encountered for interconnections within an electronic package, and that a delay of one time unit is encountered for connections between packages. Find an efficient "algorithm that will result in a network such that the maximum delay through the network is minimized."[1]

C. THE GOALS OF THIS PAPER

Lawler et al [Ref. 12] discovered efficient, easily applied algorithms to produce a solution of the partition problem for the case in which the digital circuit is in the

---

[1] Lawler, E. L., Levitt, K. N., and Turner, J., "Module Clustering to Minimize Delay in Digital Networks," IEEE Transactions on Computers, v. C-18, p. 48, January 1969.

form of a tree, and also for the more general case in which the digital circuit is in the form of a directed graph without cycles, i.e., without feedback.

The general problem, in which the digital circuit can have feedback, is considered in this paper. First a new method of representing a digital circuit in terms of a bipartite graph, due to Kodres [Ref. 10], is introduced. Then a method of reducing the general partition problem to one which can utilize Lawler's results is described. Finally a description of the general procedure in terms of the bipartite representation is given.

To this end, Chapter II contains the preliminary concepts pertinent to the problem. The formal definition of the problem and the solution are given in Sections III, IV, and V.

II. APPLICATION OF GRAPHY THEORY

Modern computer circuitry is produced using the technologies of printed circuits and integrated circuits. Both these forms of manufacturing restrict the physical interconnection of elements to a planar surface. This construction suggests the use of graph-like structures which are embedded in the plane to study the physical properties of the digital circuits.

In this section is presented a basic description of graph theory relevant to the representation and analysis of these physical properties. The terminology and definitions of graph theory vary greatly among authors. The terminology and definitions in this paper are heavily influenced by Busacker and Saaty [Ref. 3] and by Berge [Ref. 1].

A. BASIC CONCEPTS IN GRAPH THEORY

1. Definition of Undirected and Directed Graphs

In order to define an undirected graph, the concept of an unordered product of a set with itself is introduced. The symbol S & S denotes the unordered product of a set S with itself, and is defined to be the set of all unordered pairs (s&t), where $s \varepsilon S$ and $t \varepsilon S$. The symbols (s&t) and (t&s) denote the same element in S & S.

An undirected graph $G=(V,E,\Phi)$ is defined to be a nonempty set of vertices (or nodes) V, together with a set of edges E disjoint from V, and an incidence mapping Φ of E into V & V. Figure 1 depicts an undirected graph. In this

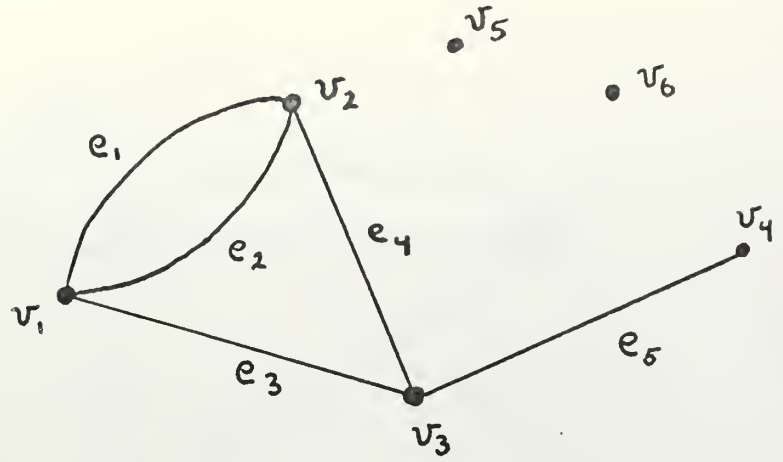

Figure 1

example V= { v_1,v_2,v_3,v_4,v_5,v_6 } and E= { e_1,e_2,e_3,e_4,e_5 }. $\Phi(e_1)=(v_1\&v_2)$ and edge e_1 is said to be incident to verticies v_1 and v_2.

Although undirected graphs are used in the analysis of digital circuits, the concept of a directed graph is more important to the representation of digital circuits. A directed graph is similar to an undirected graph but with the added notion of direction assigned to its edges. Formally, a directed graph $G=(V,A,\Delta)$ is a nonempty set of verticies (or nodes) V, together with a set of arcs A, and a directed incidence mapping Δ of A into V x V. Here V x V denotes the set of ordered pairs of V using the conventional set-theory cross product notation, and if for an arc a in A and a pair of verticies v_1 and v_2 in V, $\Delta(a)=(v_1,v_2)$ arc a is said to join its initial vertex v_1 to its terminal vertex v_2.

A directed graph is illustrated in Fig. 2. Here an arrowhead is appended to an arc to indicate the notion of direction. In this example $\Delta(a_1)=(v_1,v_2)$, and the arrow points from the initial vertex v_1 to the terminal vertex v_2.

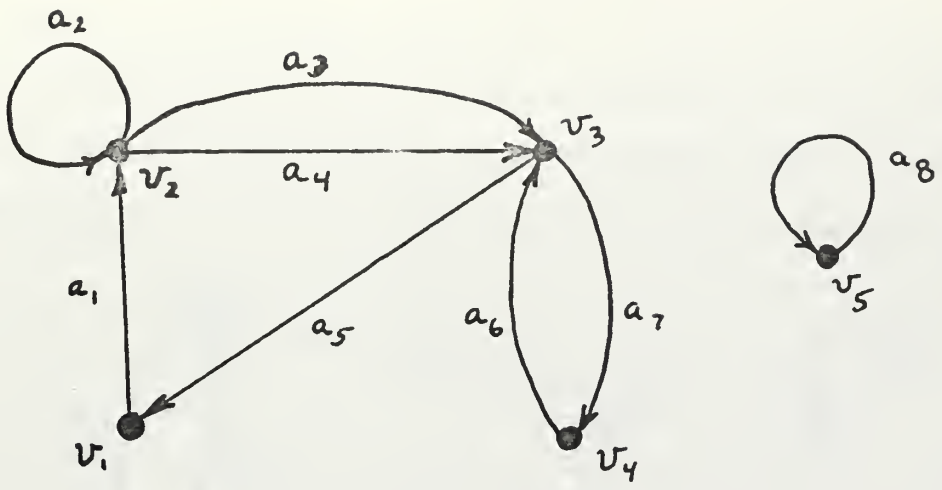

Figure 2

The symbol $a_1 \simeq (v_1, v_2)$ is read "arc a_1 joins v_1 to v_2." Similarly other terms might be used to describe this relationship such as arc a_1 "leaves" or "issues from" vertex v_1 and "enters" or "is directed towards" vertex v_2. A directed graph $G=(V,A,\Delta)$ is usually specified by $G=(V,A)$ when the incidence mapping is implicit in a description of the arcs.

There are many definitions and concepts associated with directed graphs that are similar to concepts of undirected graphs. Because the physical structure of a digital circuit can be easily represented in terms of directed graphs, the remainder of this section will deal with directed graphs. When necessary in subsequent sections, undirected terminology will be clarified. For convenience, directed graphs will be referred to in Harary's [Ref. 7] terminology as digraphs.

2. The Structure of a Digraph

Certain terminology is useful in describing the structure of a digraph. If $a_1 \simeq (v,w)$ and $a_2 \simeq (v,w)$ then arcs a_1 and

a_2 are said to be strictly parallel. If $a_3 \simeq (w,v)$ then a_1 and a_3 are parallel but not strictly parallel. The digraph in Fig. 3 demonstrates these and subsequent concepts which are introduced. In Fig. 3 arcs a_1 and a_2 are parallel.

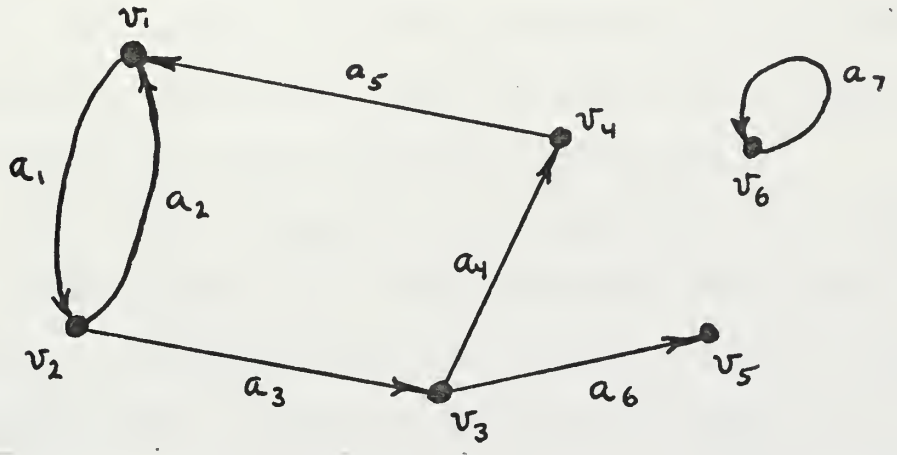

Figure 3

If G=(V,A) is a digraph, then a subgraph of G is a graph $G_1 \simeq (V_1, A_1)$ which satisfies the following conditions:

1. $V_1 \subseteq V$ and $A_1 \subseteq A$.
2. If $a \in A_1$ and $a \simeq (v,w)$ then $v \in V_1$ and $w \in V_1$.

Connectivity of a graph is defined in the undirected sense. If G=(V,E) is an undirected graph then a chain of G is a sequence of edges $e_1, e_2, \cdots, e_k$ in E together with their endpoints in V such that each edge e_i has one vertex in common with the succeeding edge e_{i+1}, and the other vertex in common with the preceding edge e_{i-1}. G is said to be connected if there exists a chain between every pair of distinct vertices. A digraph is said to be connected if its associated undirected graph is connected.

Most of this paper will be concerned with connected digraphs. A graph that is not connected can be partitioned into maximal connected subgraphs called components. In Fig. 3 vertices v_1, v_2, v_3 and v_4 together with arcs a_1, a_2, a_3, a_4 and a_5 form one component, and v_6 together with a_7 forms another component. Any result that is proved for a connected graph can be applied to each component individually.

If G=(V,A) is a digraph, then a path of G is a sequence of distinct arcs together with their endpoints such that the terminal vertex of each arc coincides with the initial vertex of the succeeding arc. A path is said to be simple if it includes no vertex more than once. In Fig. 3, the sequence $v_1,a_1,v_2,a_3,v_3,a_6,v_5$ forms a path from v_1 to v_5.

A digraph is said to be strongly connected if, for every pair of distinct vertices v and w, there is a path from v to w and a path from w to v. Therefore a strongly connected graph is necessarily connected, but not the converse.

A cycle of a digraph is a path in which the initial vertex coincides with the terminal vertex. If a cycle C traverses vertices $v_0,v_1,v_2,\cdots,v_n,v_0$ then C is said to be simple if $v_i \neq v_j$ for all $i,j=0,\cdots,n$. Again referencing Fig. 3, arcs a_1,a_3,a_4 and a_5 form a simple cycle.

An arc a is said to be positively incident with its initial vertex v and negatively incident with its terminal vertex w. The positive degree of v, $\delta^+(v)$, is the number of arcs positively incident with v, and the negative degree,

$\delta^-(v)$, is similarly defined. The degree of v, $\delta(v)$, is the sum $\delta^+(v) + \delta^-(v)$.

B. USE OF BIPARTITE GRAPHS TO REPRESENT DIGITAL CIRCUITS

The concept of a directed graph can be used to represent any circuit that is made up of interconnected functional electronic components. The circuit that is to be represented here is the digital circuit made up of interconnected fundamental logic set elements that have been previously described.

A graph with directed edges (arcs) is used to represent a circuit because it is often important to distinguish between inputs and outputs. An arc directed towards a vertex represents an input to the corresponding functional element, and similarly one directed away from a functional element represents an output.

Kodres [Ref. 10] warns, however, that it is inaccurate to use a digraph to represent a circuit by associating logic elements with vertices and interconnections with arcs. Following this practice of letting an arc represent the connection of distinct logic elements there is no way to accurately represent the branching of one signal to several logic elements.

Kodres' solution to this problem is to think of both logic elements and signals as nodes, and to represent the circuit with a bipartite graph which he calls a bi-digraph. A digraph is said to be bipartite if its vertices can be partitioned into two disjoint sets V_1 and V_2 in such a way that each arc has its initial vertex in one of the sets V_1 or V_2, and its terminal vertex in the other set. Since logic

elements are connected with interposing signals such a partition exists, and a digital circuit can be represented as illustrated in Fig. 4. Figure 4a is a simple digital circuit, and Fig. 4b is its corresponding bi-digraph representation.

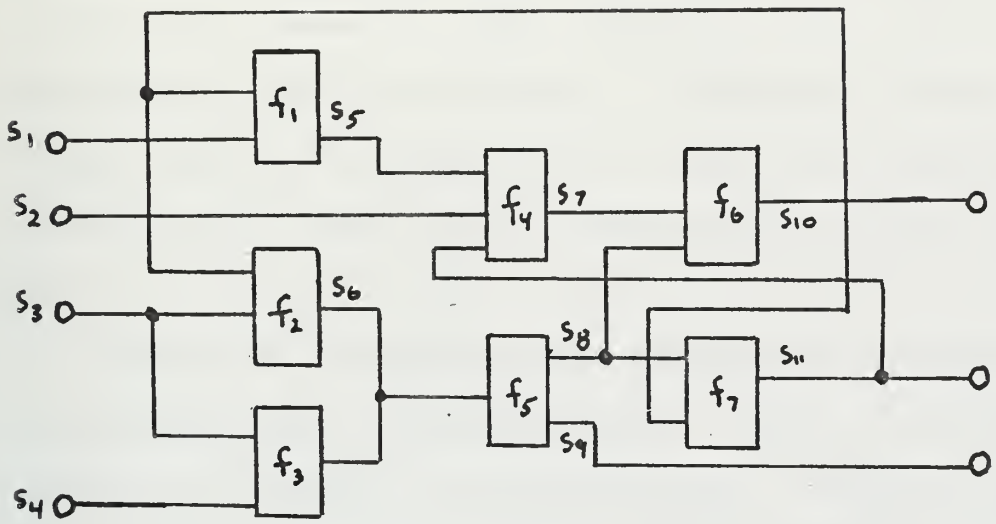

Figure 4a

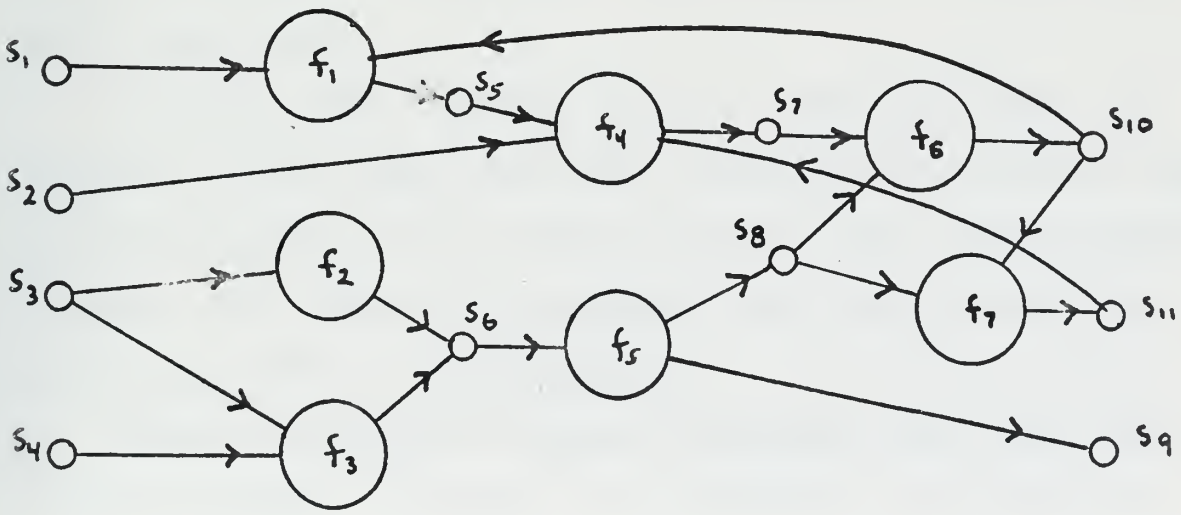

Figure 4b

The vertices of the bi-digraph corresponding to functional logic elements of the digital circuit are called functional nodes of the bi-digraph. A signal between functional nodes is, however, a more complicated notion. One must be careful to note the differences between a signal node of the bi-digraph and a signal of the bi-digraph. A signal between two functional elements is the combination of a signal node together with the arcs incident to the node. In Fig. 4b, signal node s_8 together with the output arc from f_5 to s_8 and the input arc from s_8 to f_6 constitutes a signal from f_5 to f_6.

Figure 4b illustrates many features of the bi-digraph associated with a digital circuit. First it is important to note that although the bi-digraph has a bipartite structure, it is still a directed graph, and all of the relationships in the structure of a general digraph apply equally as well to a bi-digraph. But the characteristics of digital circuits are such that the bi-digraphs under analysis here are somewhat simplified.

It can be seen from Fig. 4b that functional nodes are internal in the sense that every functional node has at least one input signal and one output signal. This does not imply, however, that there are necessarily more signal nodes than functional nodes in a bi-digraph.

Since an electronic package interfaces with other electronic packages via signals, the electronic connections to

the circuit represented by a bi-digraph are at signal nodes called input and output nodes. Input nodes are usually placed at the left of a bi-digraph, and output nodes at the right. In Fig. 4b vertex s_{10} is seen to be both an output signal node and an internal signal node.

Because of the bipartite nature of a bi-digraph, there could never be a loop from and to the same functional element. All cycles must be composed of at least two arcs. Similarly there could never be two parallel or strictly parallel arcs. Parallel arcs represent distinct signals, but in the bi-digraph, distinct signals would pass through distinct signal nodes. Strictly parallel arcs would imply redundant interconnections, which would be meaningless in a digital circuit.

C. MATRIX REPRESENTATION OF A DIGRAPH

There are various ways to use a matrix to represent the incidence relations of a graph's vertices and arcs. The two representations that are mentioned here are the adjacency matrix for representing undirected or directed graphs and the connection matrix for representing bi-digraphs.

Associated with each undirected or directed graph is an n x n adjacency matrix A where n is the total number of nodes in the graph. Row i of the adjacency matrix corresponds to vertex x_i of the graph and column j corresponds to vertex x_j, so that in the undirected case A(i,j) is equal to the number of edges incident with both vertex i and vertex j. For the directed graph the element A(i,j) is the number of

arcs directed from vertex i to vertex j. The adjacency matrix is illustrated for a directed graph in Fig. 5.

Associated with a bi-digraph is the n x m connection matrix C where n is the number of functional nodes and m is the number of signal nodes. The element C(i,j) of the

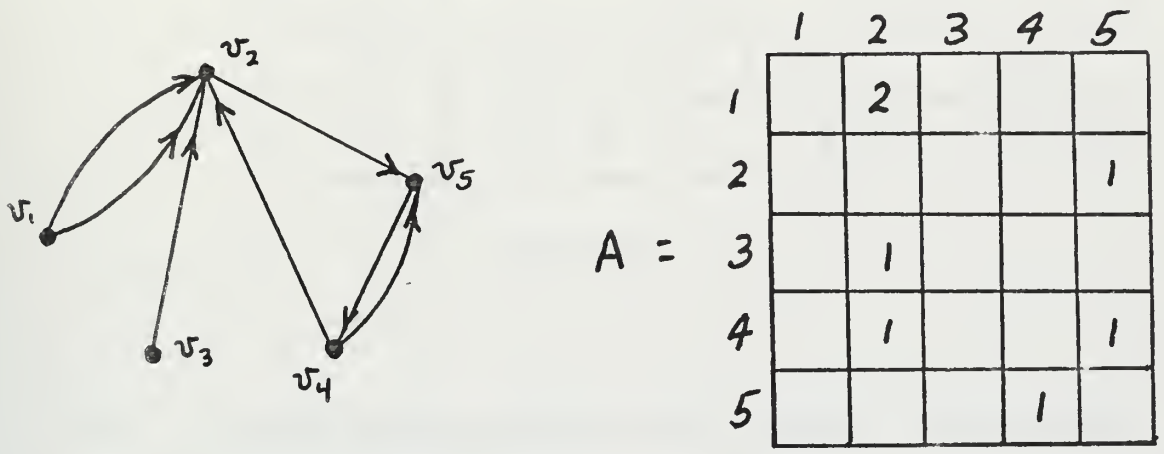

Figure 5

connection matrix is defined by Kodres [Ref. 10] as follows:

$$C(i,j) = \begin{cases} 1 \text{ if } s_j \text{ is an input signal of } f_i \\ 0 \text{ if } s_j \text{ does not touch } f_i \\ -1 \text{ if } s_j \text{ is an output signal of } f_i \end{cases}$$

Figure 6 shows the connection matrix associated with the bi-digraph of Fig. 4b.

$C =$

	1	2	3	4	5	6	7	8	9	10	11
1	1				-1					1	
2			1			-1					
3			1	1		-1					
4		1			1		-1				1
5						1		-1	-1		
6							1	1		-1	
7								1		1	-1

Figure 6

III. THE PARTITION PROBLEM WITH FEEDBACK CONSTRAINTS

A cycle of the bi-digraph corresponds to a feedback cycle in the digital circuit. Intuitively a feedback cycle is a sequence of signals and functional elements such that an electronic pulse applied at one element in the sequence could, after traversing the cycle, be detected at the element at which it was applied.

For purposes of illustration, the bi-digraph in Fig. 7 and its corresponding matrix in Fig. 6 will be used. The arcs joining the sequence of nodes f_1, s_5, f_4, s_7, f_6, s_{12}, f_1 form a feedback cycle. Cycles in the bi-digraph will henceforth be referred to as feedback cycles. Two feedback cycles are said to intersect if they share a common functional or signal node.

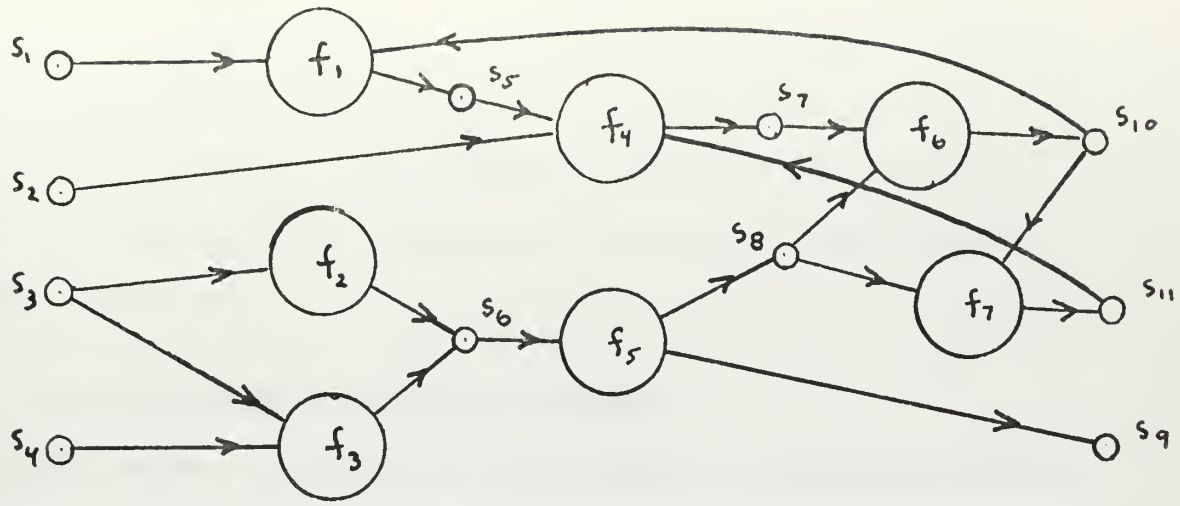

Figure 7

The problem of clustering functional elements of a digital circuit so as to minimize delay has been studied by Lawler et al [Ref. 12]. This effort considered the cases that the digital circuit was in the form of 1) a rooted tree, 2) a multi-rooted tree, and 3) a directed graph without cycles. For the first two cases computationally efficient algorithms were developed, and for the third case an algorithm was developed which minimizes delay but has certain other drawbacks. However the problem of clustering the elements of a digital circuit for the case that the circuit is in the form of a graph with cycles has not been solved.

The purpose of this section is to derive a method to reduce a digital circuit that is in the form of a cyclic graph to one that is in a form that can be processed by Lawler's algorithm. The approach taken here with respect to the description of a digital circuit is different from that of

Lawler's in that a digital circuit in this paper is represented as a bi-digraph. This will become more evident in Section IV in which the results of this section will be applied to provide an extension of Lawler's algorithm to cyclic networks.

A. CLUSTERING THE FUNCTIONAL ELEMENTS

This paper is primarily interested in determing which functional elements of the bi-digraph (those which represent elementary logic set elements) should be clustered into individual electronic packages. It would be helpful at this point to make more precise what is meant by clustering or coalescing the functional nodes of the bi-digraph.

Given the results of an algorithm that indicates which functional elements should be coalesced into a package, the bi-digraph can then be changed to represent the digital circuit in its clustered form. If, for example, functional elements f_2, f_3 and f_5 of Fig. 7 are to be coalesced, the resulting bi-digraph is shown in Fig. 8. The coalesced nodes are represented as being part of a single functional element which has been referred to as a "super-node," and which is labeled by convention with the lowest label of the coalesced nodes that it represents.

Since it is desirable to maintain the bi-digraph's bipartite structure, some clarification is needed on how it is determined which signals are made to be internal to the super-node, and which are made to be external. If a signal node

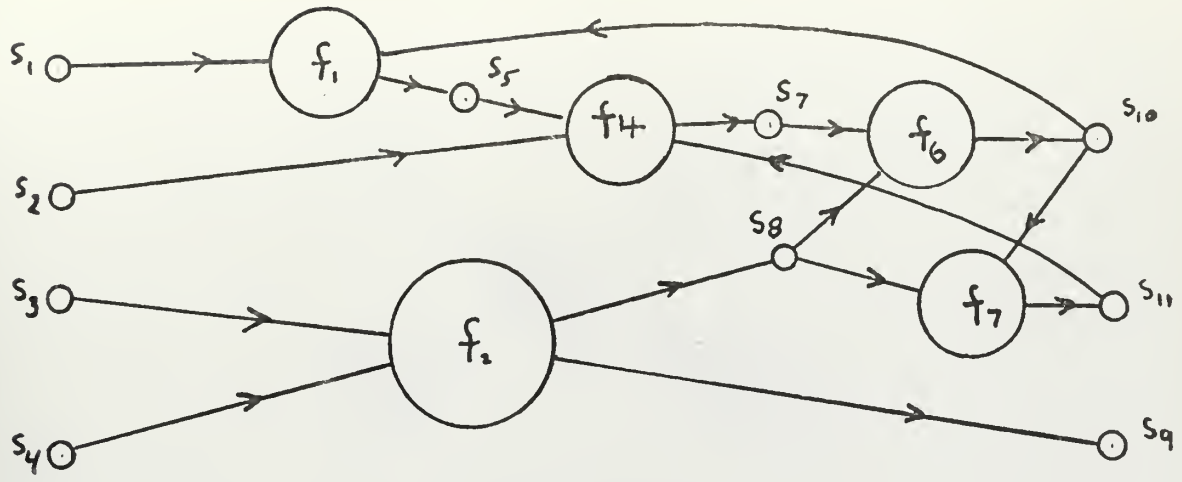

Figure 8

is adjacent only with functional nodes that are to be coalesced, then the signal node is internal to the circuit represented by the resulting super-node. If a signal node is part of a signal which communicates between a functional element to be coalesced and functional elements which are not then a signal node is similarly constructed with an arc that represents an input or output to the supernode in the coalesced version of the bi-digraph. This case is illustrated in Fig. 8 in which s_3 and s_4 are input signals and s_8 and s_9 are output signals to and from the resulting super-node. It is possible, however, that a signal node is both input and output to the functional nodes which are to be coalesced in the manner represented in Fig. 9 by node s_1. Suppose functional nodes f_1 and f_2 are to be coalesced. In this case signal node s_1 is connected to the resulting supernode by a bi-directional arc as illustrated by an edge in Fig. 10. This suffices to accurately represent the physical structure, but does cause a

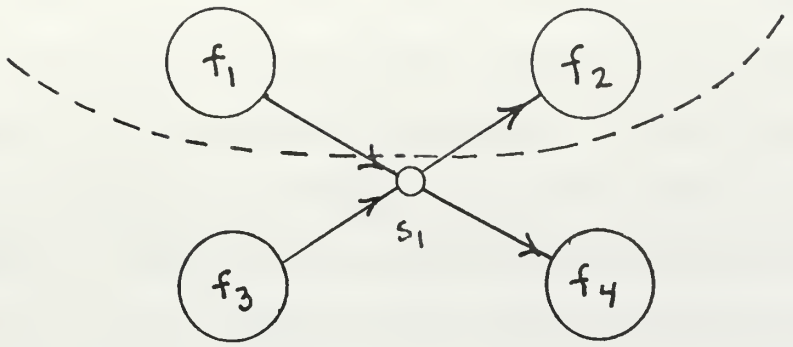

Figure 9

problem in that it introduces a bi-directional arc which must be distinguished. This can be easily done by using a special symbol in the connection matrix of the bi-digraph, and then algorithms, which are implemented on the computer, can be adjusted to treat the new arc as if it were pointing in both directions.

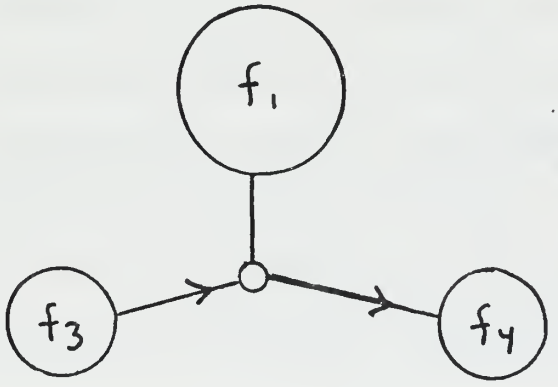

Figure 10

B. FEEDBACK CONSTRAINTS

The purpose of this section is to derive a method to reduce a bi-digraph with feedback cycles to one that is in a form that can be processed by Lawler's algorithm. To do this it is necessary to change the bi-digraph into one without feedback. This will be done by determining certain subsets of functional elements which when coalesced leave the bi-digraph free of directed cycles.

If C_i and C_k are simple cycles of a digraph then let $C_l \cap C_k$, the intersection of C_i and C_k, contain those arcs and vertices common to both C_i and C_k.

Lemma 1: Let C_i and C_k be simple cycles of a digraph. If $C_i \cap C_k$ is nonempty, then the vertices of C_i and C_k are strongly connected.

Proof: Suppose v_i is any vertex on C_i and v_k is any vertex on C_k. If either v_i or v_k is in $C_i \cap C_k$ then v_i and v_k are both in the same simple cycle, from which it follows immediately that v_i and v_k are strongly connected.

If v_i and v_k are not in $C_i \cap C_k$ then let w be a node in $C_i \cap C_k$. Since w is on C_i, there is a path in each direction between v_i and w. Also since w and v_k are both in C_k, there is a path in both directions between w and v_k. Hence there is a path in both directions between v_i and v_k containing w, and v_i and v_k are strongly connected. Proof completed.

Lemma 2: Let C_i, C_j and C_k be simple cycles of a digraph. If $C_i \cap C_k$ is nonempty and $C_k \cap C_j$ is nonempty, then the nodes of C_i and C_j are strongly connected.

Proof: Suppose that v_i and v_j are nodes on C_i and C_j, and that w_i is a node in $C_i \cap C_k$ and w_j a node in $C_j \cap C_k$. Then that part of C_i from v_i to w_i, combined with C_k from w_i to w_j, combined with C_j from w_j to v_j forms a path from v_i to v_j. A path can be similarly constructed from v_j to v_i. Hence v_i and v_j are strongly connected, and since the above can be constructed for any nodes v_i and v_j in C_i and C_j the lemma follows. Proof completed.

From the above two lemmas and the fact that strong connectivity is an equivalence relation, it is seen that strong connectivity is an equivalence relation among the vertices of a digraph that lie on at least one cycle. The equivalence classes of this equivalence relation are the vertices of the digraph that lie in strongly connected subgraphs. A lobe[2] $L(v_i)$ is a subgraph which contains exactly those vertices that are strongly connected to a vertex v_i.

The major goal of this section was to determine certain subsets of functional elements which when coalesced leave the bi-digraph free of directed cycles. The above equivalence

---

[2] This definition of a lobe is topologically equivalent to the definition of a lobe in Kamae [Ref. 9] but has been modified to suit the present discussion.

relation, when applied to the set of functional elements of the bi-digraph which lie on feedback cycles, gives a method to determine which functional elements should be coalesced. The following lemma formalizes this result.

Lemma 3: The digraph of lobes is free of cycles.

Proof: Suppose $L(v_1)$ and $L(v_2)$ are distinct lobes and that there exists a cycle passing through $L(v_1)$ and $L(v_2)$. Then the two vertices v_1 and v_2 are strongly connected. Therefore $L(v_1)=L(v_2)$ which contradicts the assumption that $L(v_1)$ and $L(v_2)$ were distinct. Proof completed.

What follows is the development of a computationally efficient method to generate the lobes of the bi-digraph represented by the matrix C in Figure 6.

C. REDUCTION OF THE PROBLEM

The first step of the algorithm is to reduce the problem of finding the lobes of the bi-digraph to one of finding the lobes of a corresponding digraph. From the bi-digraph, written in terms of its n x m connection matrix C, is produced the n x n matrix G. The rows and columns of G correspond to the functional elements of the bi-digraph (the rows of C). G is produced in such a way that the vertices of G that are in a common lobe correspond to the functional elements of C that are in a common lobe.

To do this the n x m matrices C_{-1} amd C_1 are defined. The rows and columns are labeled the same as C and the elements are defined as follows:

$C_{-1}(i,j)=1$ if and only if $C(i,j) = -1$

$C_1(i,j)=1$ if and only if $C(i,j) = 1$

Then G is formed by the following matrix product:

$$G = C_{-1} \cdot C_1^t$$

where $\cdot$ is matrix multiplication, using boolean addition.

Lemma 4: $G(i,j)$ indicates the existence of a signal from f_i to f_j in the bi-digraph.

Proof: $G(i,j) = \overset{b}{\underset{k=1}{\sum^{m}}} C_{-1}(i,k)\ C_1(k,j)$ where $\overset{b}{\sum}$ indicates boolean summation. $C_{-1}(i,j) = 1$ if and only if there exists an arc from functional element f_i to signal node s_k, and $C_1(k,j) = 1$ if and only if there exists an arc from s_k to f_j. Hence $G(i,j) = 1$ indicates that there exists a signal from f_i to f_j via some signal node s_k, $k = 1,\cdots,m$. Proof completed.

The matrix G for the graph in Figure 7 is shown in Figure 11.

$G = C_{-1} \cdot C_1^t =$

	1	2	3	4	5	6	7
1				1			
2					1		
3					1		
4						1	
5						1	1
6	1						1
7				1			

Figure 11

G can be considered as an adjacency matrix of a digraph whose vertices are the functional elements of the original bi-digraph, as shown in Fig. 12. The vertices of G will also be referred to as functional elements.

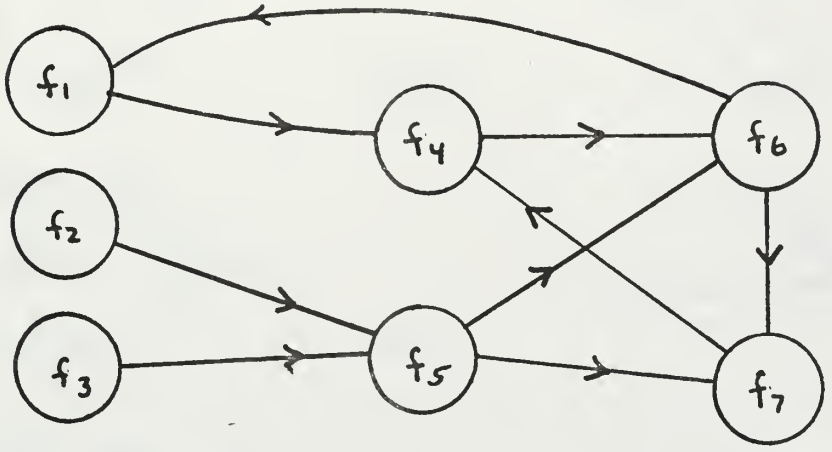

Figure 12

By the construction of G there is obviously a close relationship between the feedback cycles of C and the cycles of G. It will be shown that this relationship is such that the functional elements that lie in a lobe of C are also in a lobe of G.

There are five representative ways in which two simple feedback cycles C_i and C_j might intersect in C, as illustrated in Fig. 13. If, as shown in Fig. 13(a), C_i and C_j share a common signal, then the corresponding cycles G_i and G_j of G will share a common arc. This is by the fact that there exists an arc from f_i to f_j if any only if there exists a signal from f_i to f_j in C.

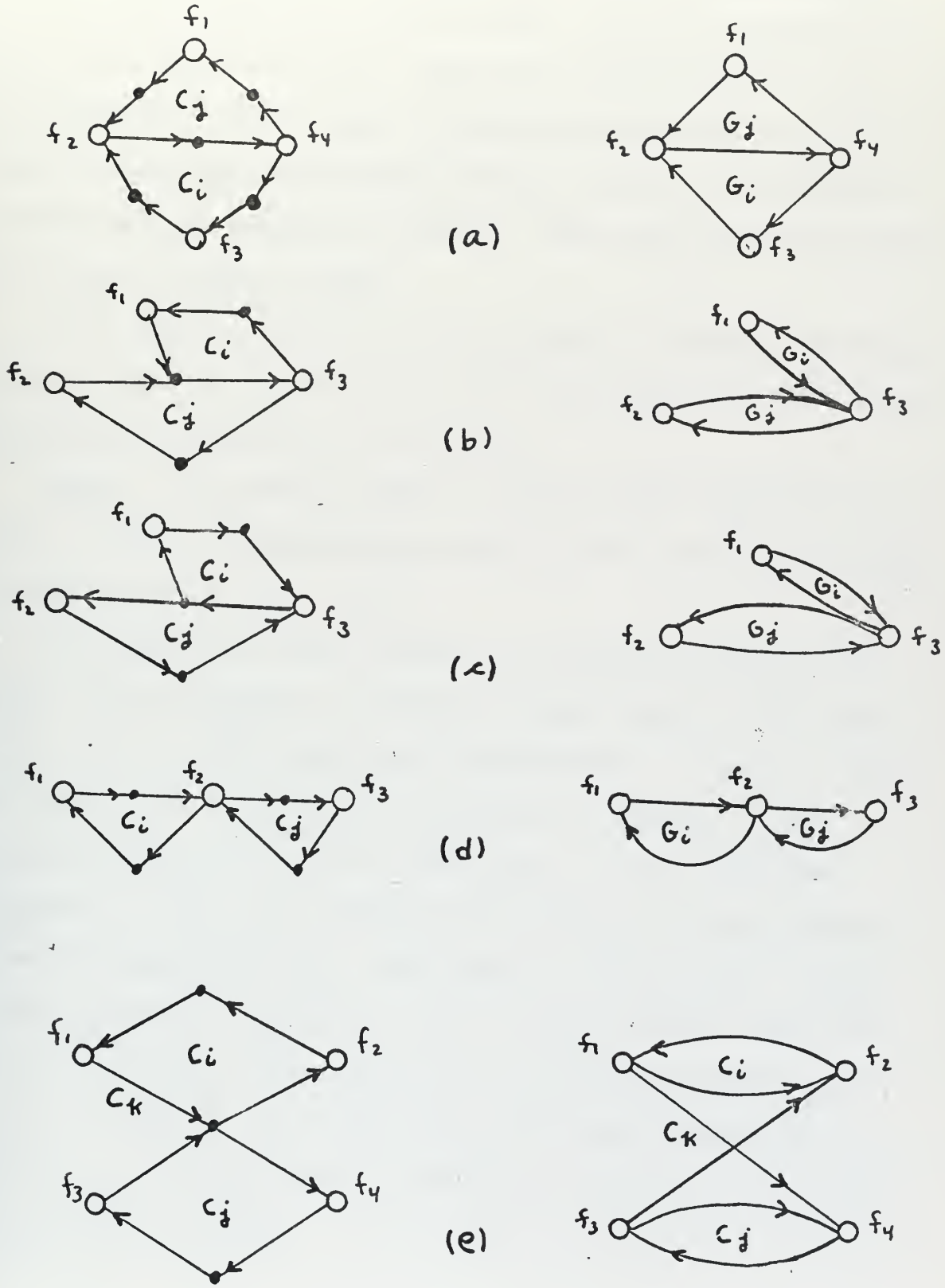
f1
Cj
f2
f4
Ci
f3
(a)
f1
Gj
f2
f4
Gi
f3
f1
Ci
f2
f3
Cj
(b)
f1
Gi
f2
Gj
f3
f1
Ci
f2
f3
Cj
(c)
f1
Gi
f2
Gj
f3
f1
f2
f3
Ci
Cj
(d)
f1
f2
f3
Gi
Gj
f1
Ci
f2
CK
f3
Cj
f4
(e)
f1
Ci
f2
CK
f3
Cj
f4

Figure 13

If C_1 and C_2 share a common input arc or output arc in C, but do not share the entire signal, then G is constructed as in Fig. 13(b) or 13(c). Since the definition of a signal includes a signal node in combination with an input arc and an output arc, the signal between f_2 and f_3 is distinct from the signal between f_1 and f_3. Hence the resulting cycles in G share a common vertex.

If as in Fig. 13(d) C_i and C_j traverse a common functional node but do not otherwise intersect, then the structure of the corresponding cycles in G is similar.

Lastly, if C_i and C_j share a common signal node in C, then a number of intersecting simple cycles occur in G. This is illustrated in Fig. 13(e).

The above five cases exhaust the possible ways in which cycles might intersect or traverse common nodes in C in the sense that any more complicated intersection of cycles could be broken down to multiple occurrences of these. By the resulting structure of G in each case, if two functional elements f_i and f_j are strongly connected in C then they are strongly connected in G. The reverse is also true. Namely, if two functional elements f_i and f_j are strongly connected in G, then there exists paths from f_i to f_j and from f_j to f_i in G, which by the construction of G means there exists signals in both directions connecting f_i and f_j in C. Hence this theorem follows.

Theorem 1. For any two functional elements f_1 and f_2, the condition that f_1 is in $L(f_2)$ holds in C if and only if it holds in G.

Therefore by finding the lobes of G one determines which functional elements of the bi-digraph represented by C should be coalesced so as to leave C free of feedback cycles.

D. GENERATION OF THE LOBES

An easy way of generating the lobes of G is by first generating the simple cycles of G. This can be done rather quickly due to an algorithm developed by Cochrane [Ref. 5].

Let $P = \{ C_1, C_2, \cdots C_k \}$ be a maximal set of simple cycles of G that is formed in the following way. First let P contain any simple cycle of G. Then successively place into P all simple cycles which intersect with any cycle previously placed into P. P is maximal in the sense that P contains all simple cycles that intersect with any cycle in P.

Let $C_1 \cup C_2$ denote the strongly connected subgraph that is formed by the arcs and vertices of C_1 and C_2. Then by Lemma 1 and Lemma 2, $\cup P = C_1 \cup C_2 \cup C_3 \cdots \cup C_k$ is a lobe of G. All of the lobes of G can be determined by finding $\cup P_i$ for each such set of simple cycles P_i in G.

The algorithm due to Cochrane to determine the simple cycles of a graph G is based on the concept of an attainability tree. The attainability tree $T(v_i)$ of a vertex v_i in G is a graph that is produced in the following way. First place v_i in $T(v_i)$. Then add to $T(v_i)$ all those nodes w_j of G that

can be reached from v_i by a path of length 1, and connect each w_j to v_i by an arc directed toward v_i in $T(v_i)$. Then again determine in G which nodes are reachable from v_i by directed paths, this time of length 2. These vertices are the same as those reachable from each w_j by paths of length 1. Place these vertices in $T(v_i)$ with arcs directed toward the corresponding w_j. Continuing in this manner for successively longer paths, the attainability tree $T(v_i)$ is produced with v_i at the root, and the branches directed toward v_i. By traveling from a node of a branch of $T(v_i)$ to the root, one traverses in reverse the nodes of a path from v_i in G. A branch of the attainability tree is terminated after the first occurrence of a node being repeated in that branch, which indicates that a simple cycle has been found.

Using Fig. 12 as an example, the attainability tree $T(f_3)$ of G, after all paths of length 4 from f_3 have been found, is illustrated in Fig. 14. On the next iteration,

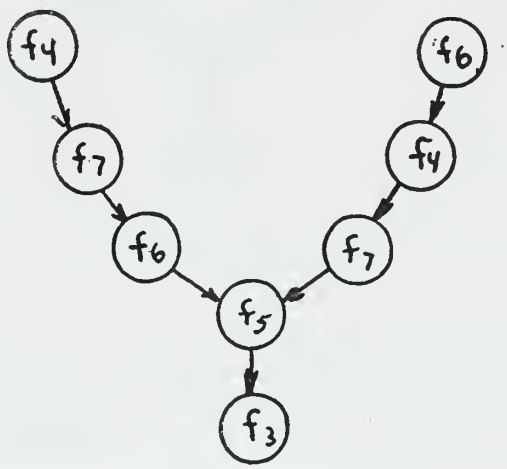

Figure 14

when considering paths of length 5 in G from f_3 the vertex f_7 is repeated in the right branch. At this point the presence of the simple cycle that traverses vertices f_7, f_4, and f_6 is discovered, and that branch can be terminated in the attainability tree. Hence, after this iteration the attainability tree is as in Fig. 15.

By continuing to build the attainability tree $T(f_3)$, all simple cycles of G will be discovered that are reachable from f_3. The cycles are simple because they are discovered at the first occurrence of a repeated vertex.

As can be seen in Fig. 15, the left branch of $T(f_3)$ will repeat the simple cycle f_7, f_4 f_6 on the next iteration;

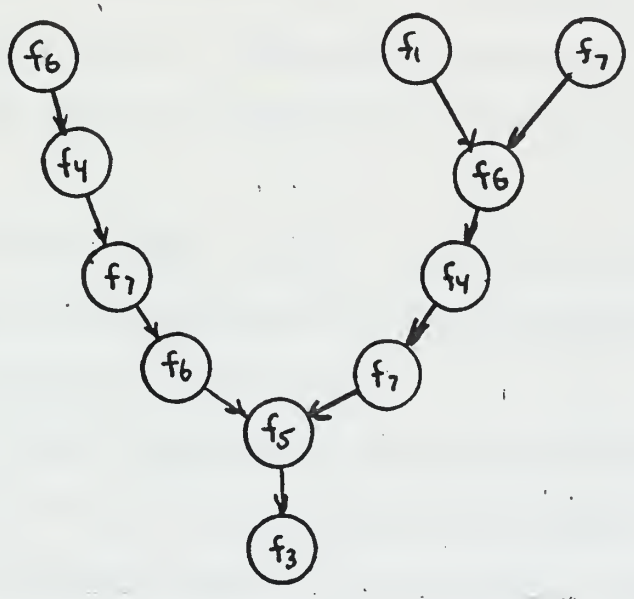

Figure 15

hence the attainability tree as constructed can repeat simple cycles. However, by first successively removing all source and sink nodes from G, the multiple occurrence of

simple cycles in an attainability tree is greatly reduced. In order to produce the simple cycles of the entire graph, it is only necessary to produce attainability trees for vertices that have not appeared in previously produced attainability trees.

Summarizing what has been said, the following is a description of the algorithm to find the simple cycles of G:

Step 1. Successively remove all source and sink nodes of G. End Step 1.

Step 2. Produce the attainability tree $T(f_i)$ for some node f_i that has not previously appeared in an attainability tree. Each time a vertex is added to the tree, check the vertices of its branch for the duplication of a vertex, which would indicate a simple cycle. Terminate a branch of the tree as soon as a simple cycle is indicated in the branch. End Step 2.

Step 3. Record the simple cycles. If there are any vertices of G that have not yet appeared in an attainability tree, then return to Step 2. End Step 3.

The goal of this section was to produce an algorithm to find the lobes of G, and thereby indicate which functional nodes of the bi-digraph should be coalesced. By adding the following two steps to the above algorithm, this has been accomplished.

Step 0. From the connection matrix C of the bi-digraph, compute $G = C_{-1} \cdot C_1^t$. Proceed to Step 1. End Step 0.

Step 4. From the list of simple cycles from Step 3, determine a maximal set of intersecting simple cycles $P_i = \{ C_1, C_2, \cdots C_k \}$. Find a lobe of G by computing $\cup P_i$. Repeat this step until all such sequences of cycles have been processed.

E. COMMENTS ON THE IMPLEMENTATION OF THE ALGORITHM ON THE COMPUTER

In terms of the conservation of storage, it is not efficient to store a graph in the computer in matrix form. In particular, the connection matrix of the bi-digraph and the adjacency matrix of the digraph that are used in the present algorithm are sparse matrices. Although there are several methods of storing a sparse matrix compactly, some inhibit the quick retrieval of information by requiring a program to perform several computations in order to determine the value of a particular entry.

In the computer program which illustrates the algorithm developed here (page 68), the connection matrix of the bi-digraph has been somewhat modified. Using the connection matrix in Fig. 6, the functional nodes are numbered 1 through 7, and the signal nodes are numbered 8 through 18. Then the graph is stored in two lists. This is illustrated in Fig. 16.

The index of list N corresponds to a node of the graph, and the $i^{\underline{th}}$ entry N(i) indicates the corresponding starting index of list M. Starting at N(i), the entries of M indicate which nodes of the graph are connected from node i by a

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
	1	2	3	4	5	7	8	9	10	11	12	13	14	15	16	18	18	20

M	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	12	13	13	14	15	16	17	18	1	4	2	3	4	5	6	6	7	7	1	4

Figure 16

directed arc. The complete list of such nodes occupy $M(N(i))$, $M(N(i)+1), \cdots, M(N(i+1)-1)$ in M. For example $N(5) = 5$ and $N(6) = 7$, which means that an arc issues from functional node f_5 to the nodes listed in $M(N(5))$ and $M(N(5)+1) = M(N(6))$, i.e., to signal nodes 15 and 16 (8 and 9 in Fig. 6).

The above construction is applicable to both digraphs and bi-digraphs. In the case of bi-digraphs, as was the case in Fig. 16 for the n x m connection matrix of Fig. 6, the first n indices of list N correspond to the n functional nodes, and the last m indices of N correspond to the m signal nodes. The resulting saving in space is obvious, for to store a graph in the computer in this way, one needs only as many memory locations as there are arcs and nodes in the graph.

It is possible to store the bi-digraph even more compactly by making further use of the bi-partite structure of the graph, and indicating the direction of arcs with a + or - sign as is done in the connection matrix. To do this the first list contains indices for only the functional nodes, and the second list contains signal nodes, with the sign of the signal node indicating the direction of the arc (towards or away from the signal node). This method of storing the graph, however,

makes it difficult to determine which functional nodes are connected to a given signal node. To obtain this information, a search of the entire length of the second list is required in order to determine each occurrence of the given signal node, and therefrom to determine an immediately connected functional node.

The algorithm developed in this section is illustrated by the computer program on page 68. The program is divided into several logically independent sections. The program starts with the bi-digraph C stored in compact form. Then two list structures are built which represent the directed graph of functional nodes G, and which enable the quick removal of successive sink and source nodes. From the resulting structure of the graph the attainability trees are produced using a list structure, and the simple cycles of the graph are discovered. Finally the simple cycles are combined as previously described in order to produce the lobes of the graph. These logical divisions in the program allow considerable overlaying of data structures from one part of the program to the next to conserve storage.

IV. EXTENSION OF THE CLUSTERING ALGORITHM TO CYCLIC GRAPHS

The partition problem is far from being solved in the general case. As mentioned previously, Lawler et al [Ref. 12] have provided solutions to the partition problem for the case in which the digital circuit is in the form of a rooted tree, or in the form of a multi-rooted tree. By applying the results of Section III, the functional elements which lie on feedback cycles can be coalesced to produce a graph without feedback. If this resulting graph has the form of one of the above two types of trees, then Lawler's algorithm can be applied to complete the solution to the problem. This assumes, of course, that each lobe of the original graph satisfies the constraints on the maximum number of logic elements and external connections that can be associated with a single cluster.

If the graph which results from coalescing the lobes is not in the form of a tree, but rather is in the form of a general acyclic graph, then Lawler's extended algorithm can be used, but only by allowing undesirable node replication. This node replication renders the algorithm unsatisfactory from a practical point of view. It is included primarily for theoretical reasons.

Sections IV A, IV B and IV C briefly describe Lawler's results as applied to rooted trees, multi-rooted trees, and acyclic graphs. This presentation differs from Lawler's in that the application here is to the digital circuit expressed

as a bi-digraph, rather than a digraph, and as such several modifications have been made. Section V describes the problem of coalescing a subset of the functional elements of a lobe in the case that they do not fit within a single cluster.

A. CLUSTERING ALGORITHM APPLIED TO RESULTANT ROOTED TREES

Given a digital circuit expressed in the form of a bi-digraph, it is assumed that the algorithm developed in Section III has been applied and that the lobes of the bi-digraph, if there were any, have been coalesced into super-nodes. It is further assumed in this and in the following two sections that the lobes were within the space and external connection constraints for a single cluster.

This section considers the case that the bi-digraph of signal nodes, functional nodes, and super-nodes is in the form of a rooted tree, a directed graph with only a single source node (in-degree zero) or sink node (out-degree zero), and whose corresponding undirected graph is connected and free of undirected circuits. Functional nodes f_2 and f_5 represent super-nodes formed by coalescing lobes of functional nodes in the original bi-digraph. The graph in Fig. 17 is a sink tree (all the arcs are directed toward the root). The procedure to be described works equally as well for source trees, with certain obvious modifications.

The problem is to determine which functional elements of the bi-digraph should be coalesced so as to minimize delay

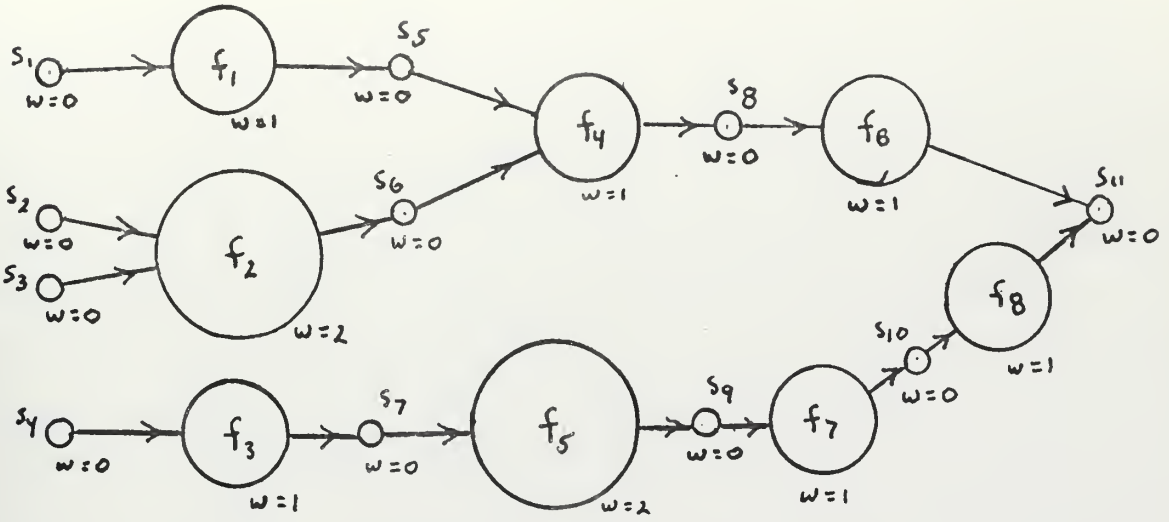

Figure 17

through the network. An external arc causes one unit of delay. Each coalesced group of functional elements must satisfy two further constraints: 1) the number of functional elements must not exceed the bound M, and 2) the number of external connections must not exceed P. This problem can be formulated as that of labeling the nodes of the graph with integers. The functional nodes of the graph to be coalesced will be those functional nodes that are connected and that are assigned the same label. The label of the functional node will indicate the maximum delay through the circuit to that point, following the convention that external signals have a delay of one time unit and internal signals have negligible delay.

The labeling process can be described as follows. Associate with each node of the graph a non-negative weight w_i

which indicates the number of functional elements the node represents. Hence signal nodes have weight zero, functional nodes of the original bi-digraph have weight one, and super-nodes have weight equal to the number of functional nodes they represent. Then label the nodes of the sink tree according to the following rules:

Rule 1. Nodes with in-degree zero (input signal nodes) will be given the label 0.

Rule 2. A node i is to be given a label at least as large as the largest label given to any of its predecessors.

Rule 3. The sum of the weights of any given node, and all of its predecessors which have the same label, is not to exceed M.

Rule 4. The total number of pin connections of a given node and all of its predecessors which have the same label is not to exceed P.

Rule 4 needs some clarification. A cluster will be composed of functional elements with the same label. Such a cluster is determined by finding a functional element no successor of which has the same label. Then, form a cluster with that element and all of its predecessors that have the same label.

An external connection is a signal that communicates between nodes of a cluster and nodes not in a cluster. As successors of a node are labeled, it must be determined that the total number of arcs that lie on external signals to the indicated cluster, is within the bound P. The indicated cluster is the cluster that will be formed by this group of

nodes with the same label. If P is exceeded, then the label assigned to any successors of the last node labeled must be labeled one greater.

It is desired to produce an algorithm that labels the nodes of the graph according to the above rules, and such that the largest label given any node is as small as possible. The following notation from Lawler will help to describe the algorithm which accomplishes the desired labeling.

For any subset of nodes S (both functional and signal) let p(S) equal the number of arcs with exactly one end point in S, and which lies on a signal that communicates between a node of S and a node not in S. Hence p(S) indicates the number of pin connections due to external signals to and from S. Let R(i,k) denote the set of k-predecessors of node i, in other words, the set of nodes which are predecessors of node i, and which have the label k. Let $w_i(k)$ be the total weight of all of the k-predecessors of node i.

Then the labeling algorithm developed by Lawler and applied to bi-digraphs in the form of a rooted tree is as follows:

Step 0. Label all input nodes 0.

Step 1. Find any unlabeled node i, all of whose predecessors have been labeled. Let k be the largest label applied to any of these predecessors. If $p(R(i,k) \cup \{i\}) \leq P$, and if $w_i^t + w_i(k) \leq M$, then assign label k to node i, else assign label k+1 to node i.

Repeat Step 1 until all nodes have been labeled.

The result of this algorithm for M = 3 and P = 5 applied to the bi-digraph of Fig. 17 is illustrated in Fig. 18. The label applied to a node is enclosed in parentheses. The resulting clusters are indicated by broken lines.

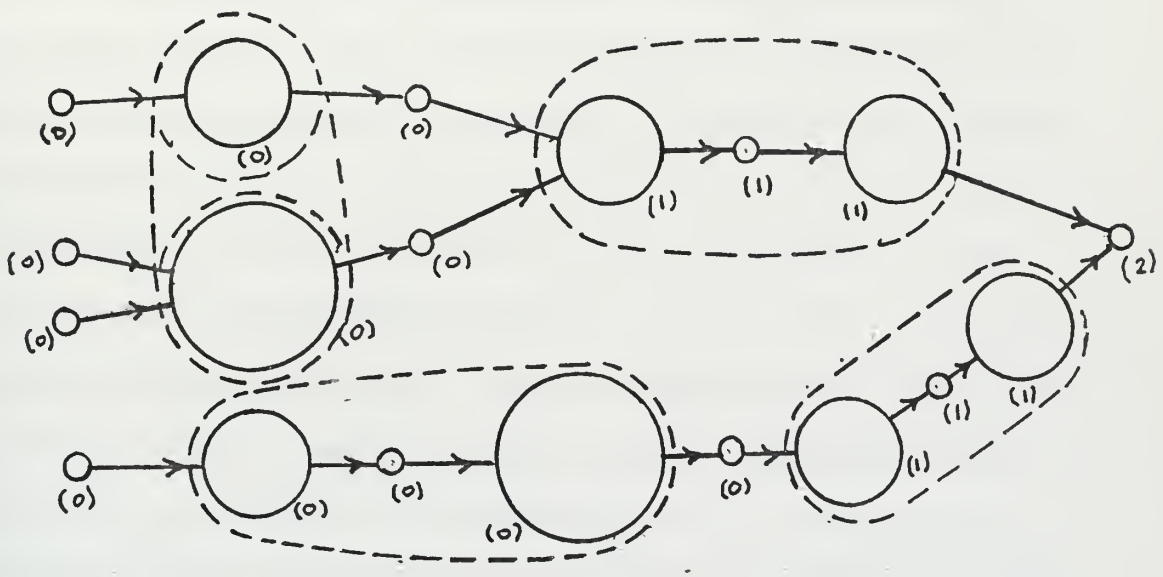

Figure 18

According to the procedure that functional elements are to be placed in the same cluster if they are connected by a path and have the same label, nodes f_1 and f_2 each form a separate cluster. It should be noted that by joining f_1 and f_2 in the same cluster, the constraints are not exceeded, and the delay through the circuit is not increased. Hence they are shown in Fig. 18 as being in the same cluster. The above algorithm works equally well for source trees as for sink trees by simply replacing each occurrence of the word "predecessor" with "successor," and each occurrence of the

word "sink" with "source," in the above discussion and algorithm statement.

In Lawler's statement of this algorithm nodes were each given a weight of one. The concept of weighted nodes was defined to facilitate the extension of this algorithm to multi-rooted trees. This will be discussed in Section IV B. However, using the weight of a node to represent the number of functional elements that a node indicates, facilitates the application of the algorithm to the bi-digraph. This makes it easy to distinguish between signal nodes, functional nodes, and super-nodes which result from previous clustering. By including signal nodes in the labeling algorithm, such definitions as the set of k-predecessors of a node can be stated without the complication of having to distinguish between different types of nodes. Since a signal node is assigned weight zero, it is not necessary to treat signal nodes separately when determining the total number of functional elements that are in a cluster. The weight of a signal node is simply added to the previous total. Similarly, since a super-node represents more than one functional element of the original bi-digraph, the total number of functional nodes that it represents is easily obtained so that constraint (1) may be checked simply by addition of weight.

B. EXTENSION OF THE ALGORITHM TO RESULTANT MULTI-ROOTED TREES

If the bi-digraph which results from the coalescing of lobes is in the form of a multi-rooted tree (a tree with more

than one source or sink node), then the multi-rooted tree can be separated into a set of rooted trees to which the algorithm of Section IV A can be applied. The bi-digraph of Fig. 19 will be used to illustrate the procedure of this section. The weights of each functional node again indicate the number of functional elements of the original bi-digraph that are represented, and the weight of each signal node is zero.

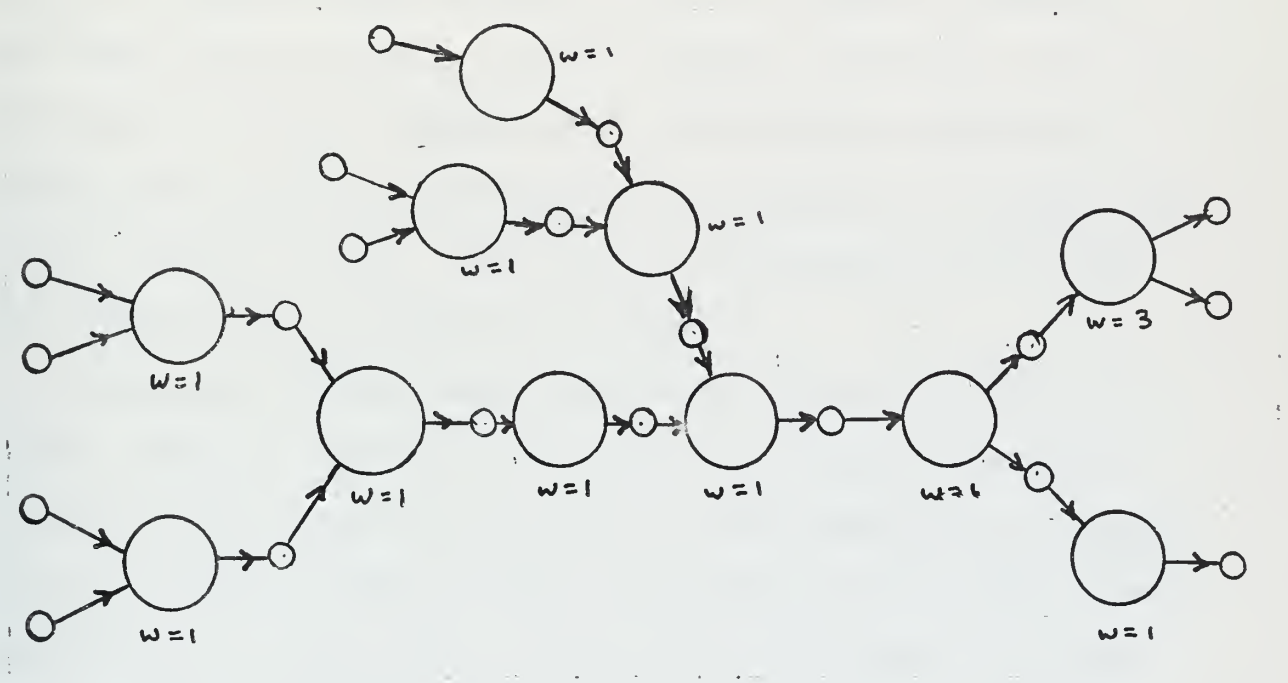

Figure 19

The first phase of the following algorithm separates the multi-rooted tree into a set of rooted trees that can be processed individually using the algorithm of Section IV A. This is done by temporarily removing certain arcs of the graph. The arcs removed are placed in a set A. The removal

of the arcs is done in such a way that each source-to-sink path in the multi-rooted tree passes through at most one of these arcs.

The tree separation procedure is expressed as Steps 0 and 1 of the clustering algorithm for multi-rooted trees.

Step 0. Choose any input signal node and all of its successors as an initial source tree. Check each of the nodes of this tree.

Step 1. Find an arc, exactly one end of which is checked, and place this arc in A. If the terminal (initial) vertex of this arc is the one checked, then the initial (terminal) vertex and all of its predecessors (successors) is chosen as a sink (source) tree. Check all of the nodes of this tree and repeat Step 1 as often as necessary.

The result of the above two steps is shown in Fig. 20. Signal node s is chosen arbitrarily as the initial source node, and the arcs that are put in A are indicated by shading. It can be seen in Fig. 20 that any source-to-sink path passes through at most one arc in set A. The bi-digraph has been separated into three rooted trees, the source tree with root s, and two sink trees.

The next step in the clustering algorithm is as follows:

Step 2. Apply to each rooted tree the labeling procedure of Section IV A. For source trees the labeling procedure goes from sinks to source, and for sink trees the procedure goes from sources to sink.

a_i in A, let $(j,k)_i$ denote the signal from f_j to f_k which includes arc a_i, where f_j and f_k are in F. Then the maximum delay along a path of the type completely contained in a single cluster is D where

$$D = \max_{f_i \in F} \{ k_i \}$$

where k_i is the label applied to functional node f_i. The maximum delay encountered along a path of the type that includes an arc of A is D' where

$$D' = \max_{all(j,\ell)_i} \{ k_j + k_\ell + 1 \}$$

where the maximum is taken over all signals $(j,\ell)_i$ associated with all arcs a_i in A, and k_j and k_ℓ are the labels assigned to functional nodes f_j and f_ℓ respectively. Care in defining D' in terms of signals instead of arcs is required by the fact that the label, assigned to a signal node, which might be one endpoint of an arc in A, might be one greater than any label assigned to the functional nodes which immediately precede it in the case of a sink tree, or which immediately succeed it in the case of a source tree. The delay which is associated with such a path is the same as the largest label assigned to a functional node of the path (not the signal node, which could have a label one greater).

If $D' \leq D$, then the clustering is optimal, since the maximum delay of the circuit is along a path that is completely contained in a single rooted tree. If on the other hand

$D' > D$, it might be possible to decrease the maximum delay by one unit.

Consider the arc $a_i \in A$, which is traversed by the path P of greatest delay. Since $a_i \in A$, arc a_i is an external arc in the clustering indicated. Arc a_i might be critical in the sense that if a_i could be made internal to a cluster along path P, the maximum delay of the circuit might be reduced by one unit. It is possible, however, that if A is changed by removing a_i and by including some other arc, that another arc might become critical. The possibilities have been enumerated and analyzed by Lawler and the results are expressed in the following steps which complete the algorithm. The steps define a relabeling of the nodes of the graph to indicate clustering with minimal delay. The steps are substantially the same as were expressed by Lawler et al [Ref. 12], except that certain definitions have been modified to account for the bi-digraph representation. Also the algorithm as stated here takes into account both the maximal weight constraint M and the external connection constraint P.

Step 3. Compute

$$D = \max_{f_i \in F} \{ k_i \}$$

and compute

$$D' = \max_{\text{all}(j,\ell)_i} \{ k_j + k_\ell + 1 \}$$

If $D' \leq D$, proceed to Step 6, else proceed to the relabeling procedure, Step 4.

Condition				Action
$k+k' > D'-1$				Go to Step 6
$k+k' = D'-1$	$w_i + w_i(k) + w_i'(k') \leq M$ and $p(\{i\} \cup R(i,k) \cup S(i,k')) \leq P$			Assign node i label k Add $w_i'(k')$ to w_i Replace $\{i\}$ by $\{i\} \cup S(i,k')$
$k+k' = D'-1$	$w_i + w_i(k) + w_i'(k') > M$ or $p(\{i\} \cup R(i,k) \cup S(i,k')) > P$			Go to Step 6
$k+k' < D'-1$	$w_i + w_i(k) \leq M$ and $p(\{i\} \cup R(i,k)) \leq P$			Assign node i label k
	$w_i + w_i(k) > M$ or $p(\{i\} \cup R(i,k)) > P$	$(k+1) + k' < D'-1$		Assign node i label $k+1$
		$(k+1) + k' = D'-1$	$w_i + w_i'(k') \leq M$ and $p(\{i\} \cup S(i,k')) \leq P$	Assign node i label $k+1$ Add $w_i'(k')$ to w_i Replace $\{i\}$ by $\{i\} \cup S(i,k')$
			$p(\{i\} \cup S(i,k')) > P$ or $w_i + w_i'(k') > M$	Go to Step 6

NOTE: Primed variables refer to successors
$w_i(k')$ denotes total weight of all the k' successors of node i
$S(i,k')$ denotes the set of k' successors external to the tree in question

Table 1

Step 4. Repeat this step for each rooted tree in turn in opposite order from that in which they were formed in Steps 0 and 1. Assume the tree in question is a sink tree (modification for a source tree is straightforward).

4a) Label all source nodes 0.

4b) Find any unlabeled node i, all of whose predecessors have been relabeled. If this node is the sink node, pass to substep 4c below. Let k be the largest label of any predecessor functional node, and k' be the largest label of any successor functional node not contained in the sink tree in question. Assign a new label to node i according to Table 1. Repeat this step as often as necessary.

4c) In the case of a root node, let k be the largest label of any predecessor functional node. If $w_i+w_i(k) \leq M$, assign the label k to the node, otherwise assign the label k+1. If the value of this label is greater than D'-1, go to Step 6.

Step 5. Replace each label k_i obtained for a source tree in Step 4 by (D'-1)-k. The resulting labeling is optimal. End of procedure.

Step 6. Replace each label k_i obtained for a source tree in Step 2 by $D-k_i$. This labeling is optimal. End of procedure.

The results of applying this algorithm to the bi-digraph of Fig. 19 and Fig. 20 will be illustrated by Fig. 21 through 23. Figure 21 shows the results of Step 2, the labeling procedure of Section IV A for M = 3 and P = 5. The clusters

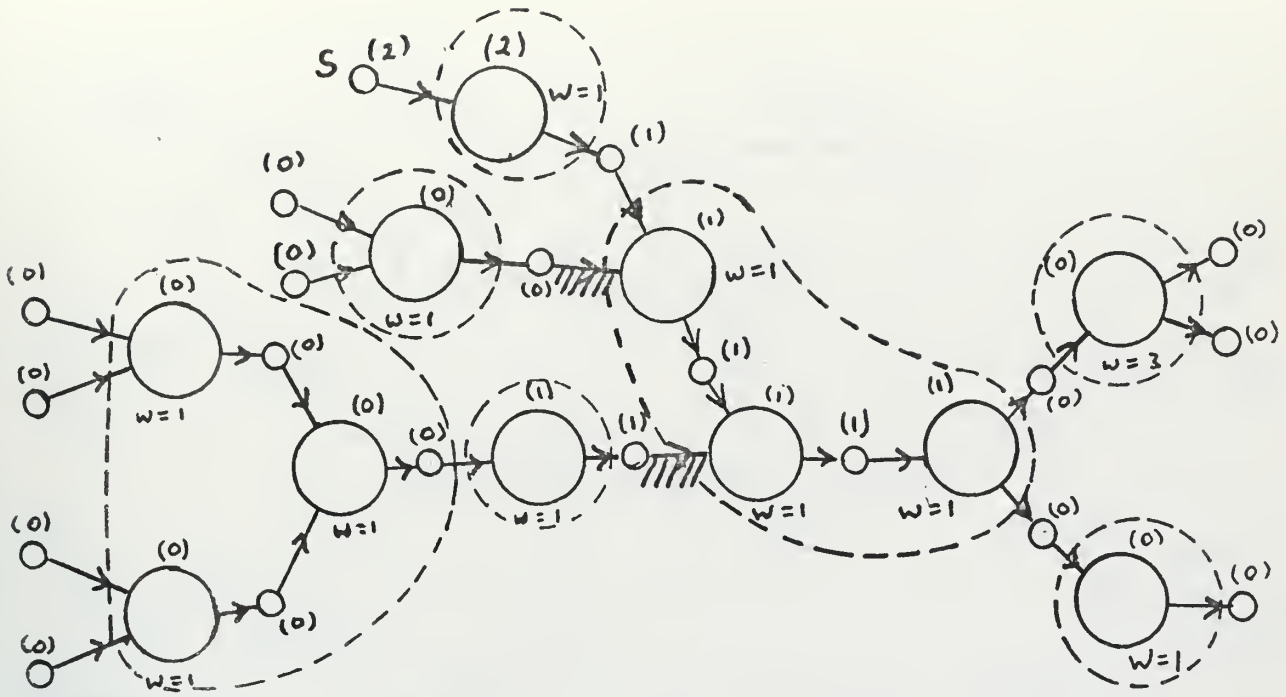

Figure 21

are indicated in Fig. 21 by sets of encircled nodes. Computing the value required in Step 3, it is found that

$$D = \max_{f_i \in F} \{ k_i \} = 2$$

and

$$D' = \max_{\text{all}(j,\ell)_i} \{ k_j + k_\ell + 1 \} = 3.$$

Since D' > D the relabeling procedure of Step 4 using the instructions of Table 1 is required to produce an optimal clustering. Following these instructions, the resulting labeling is illustrated in Fig. 22. Note that the arc which has weight 3 is a super-node and has this weight due to the

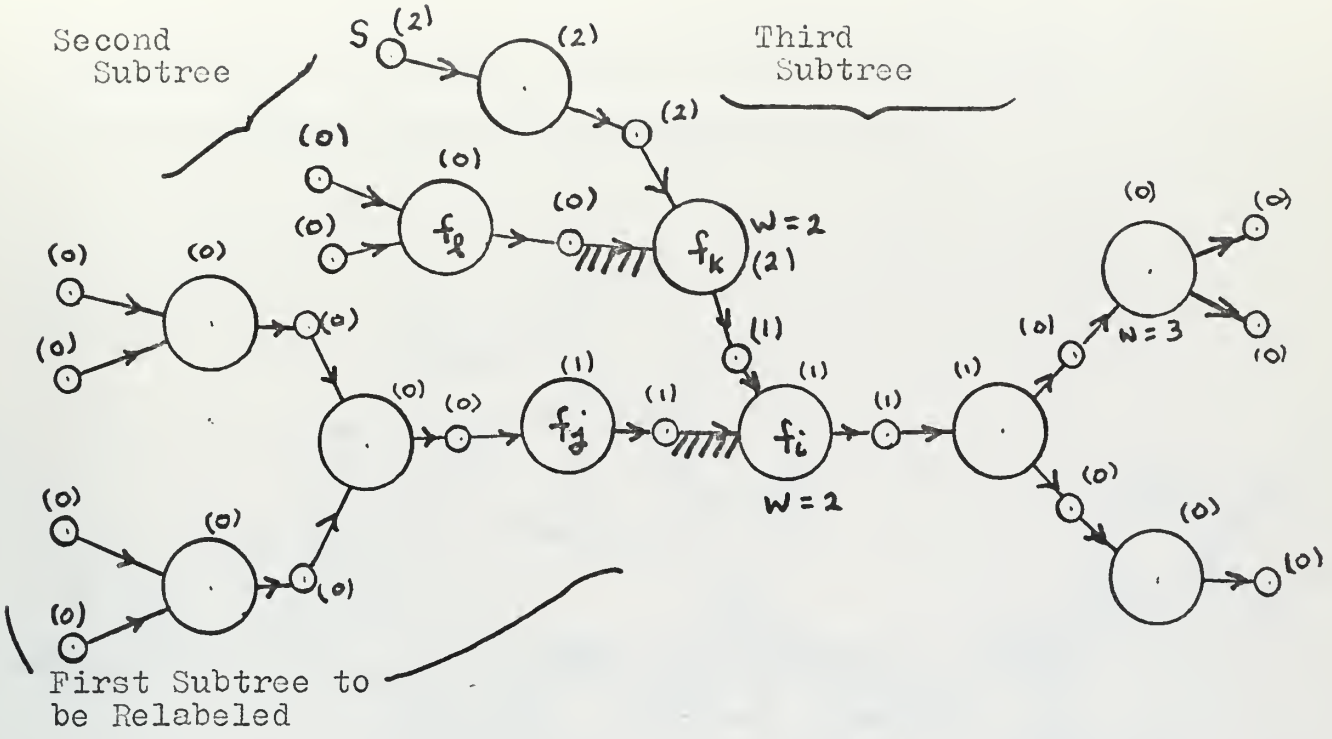

Figure 22

clustering algorithm of Section III. The functional elements labeled f_i and f_k which now each have weight 2 are not super-nodes. Their weights were originally 1, and were increased by the instruction add $w_i'(k')$ to w_i in Table 1. This is required as a result of the relabeling scheme, since node f_j will now be included in the cluster with f_i, and f_ℓ with f_k. In order to complete the relabeling algorithms for the third sub-tree, it is necessary that the weights of these additional functional elements be used in determining the total weight of the cluster.

Step 4 is now completed, and Step 5 is executed. This completes the relabeling process, and as can be seen in Fig. 23, the maximum delay has been reduced in two.

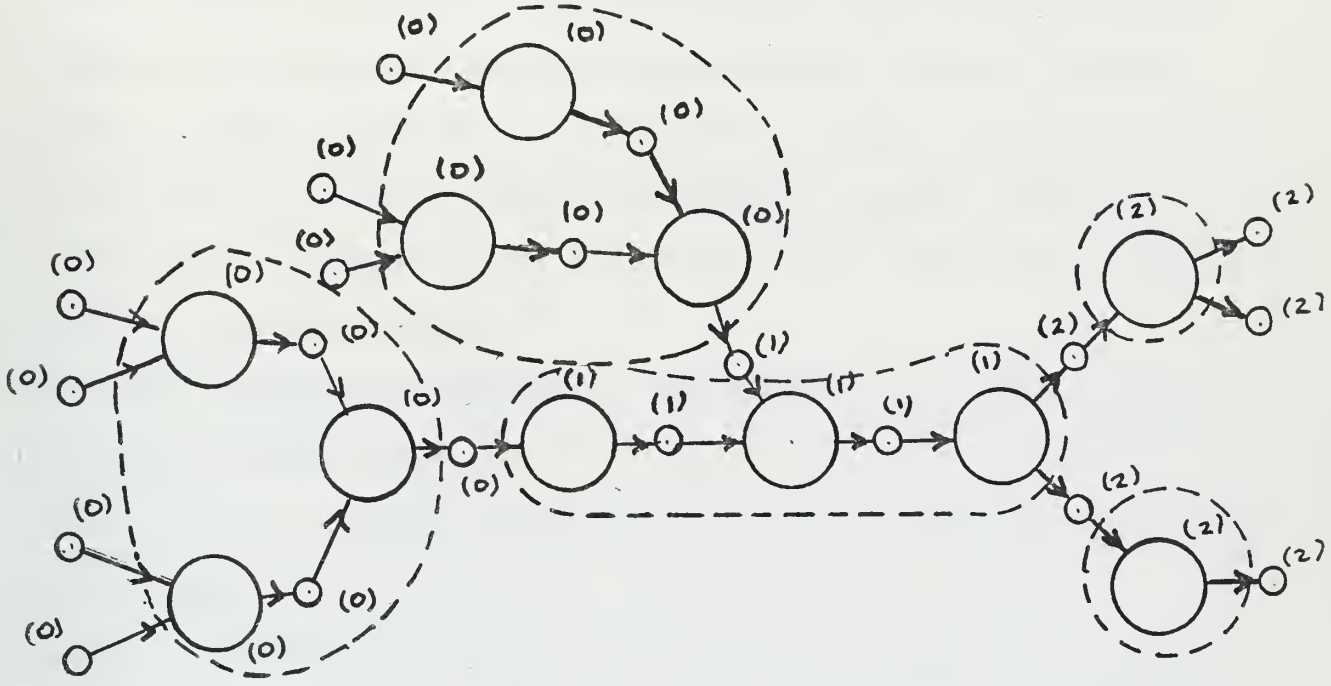

Figure 23

C. COMMENTS ON A PROCEDURE FOR RESULTANT ACYCLIC GRAPHS

Lawler's algorithm can be applied to a resultant acyclic graph that is not in the form of a tree, but only with limited success. The procedure that was applied to rooted trees can be used to produce a minimal cluster of an acyclic graph, but only with the added provision that a certain amount of node replication be allowed. Hence, if after the application of the algorithm of Section III the resultant acyclic graph

is not in the form of a tree (either rooted or multi-rooted), a satisfactory solution does not exist to complete the clustering. Although the procedure of Section III makes the algorithm applicable to a larger class of graphs, namely those graphs containing directed bycles, the clustering algorithm for resultant general acyclic graphs is still incomplete. This section will first show how the algorithm for rooted trees has been applied to acyclic graphs, and will then give an example which illustrates the reason this application is unsatisfactory.

The algorithm of Section IV is applied to acyclic graphs in the following way:

Steps 0 and 1. Label the nodes as in the procedure for rooted sink trees.

Step 2. Locate each node of the graph that has the property that no successor of the node has the same label as the node itself. This node and all of its k predecessors form a single cluster.

Step 3. Construct a new graph as follows:

a) For each node of the original graph, create a node j_q for each cluster q in which node j is contained.

b) For each arc a_i, from nodes i to j in the original graph (one of these is a signal node and the other a functional node due to the bipartite structure), create an arc from node i_p to j_q where

i) p = q if i and j have the same label

ii) p is chosen arbitrarily, otherwise.

The application of the above steps is illustrated in Fig. 24. In this example the result of Step 2 produces immediately a feasible clustering for M = 3, and in such a way that Step 3 produces an optimal clustering as is shown in Fig. 24a. Node replication is illustrated by signal node s_1 in Fig. 24b. In this case, following Step 3 precisely

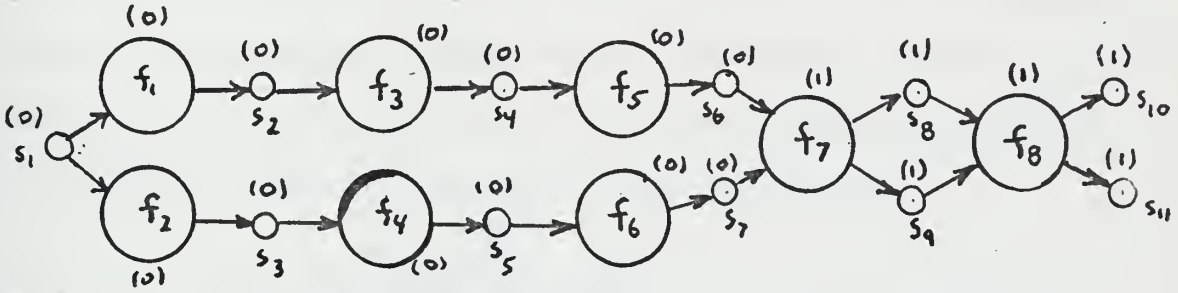

Figure 24a

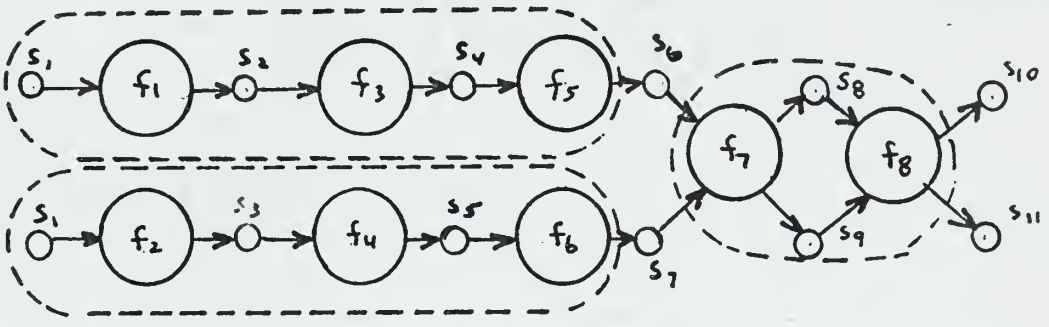

Figure 24b

as stated, signal node s_1 is replicated and made a part of each cluster indicated. This procedure is interpreted to mean that the same signal is applied at both occurrences of node s_1. In the above case this poses no problem, since by the method of clustering nodes described in Section III A, signal node s_1 would be external to both clusters, and would merely be common to the input signals to each cluster.

But in more complicated structures node replication becomes a major problem. For the graph in Fig. 25a, and the resulting structuring in Fig. 25b, excessive node replication renders the procedure impractical. Note that not only are signal nodes replicated, but so also are functional nodes. This usually requires extensive logic redesign.

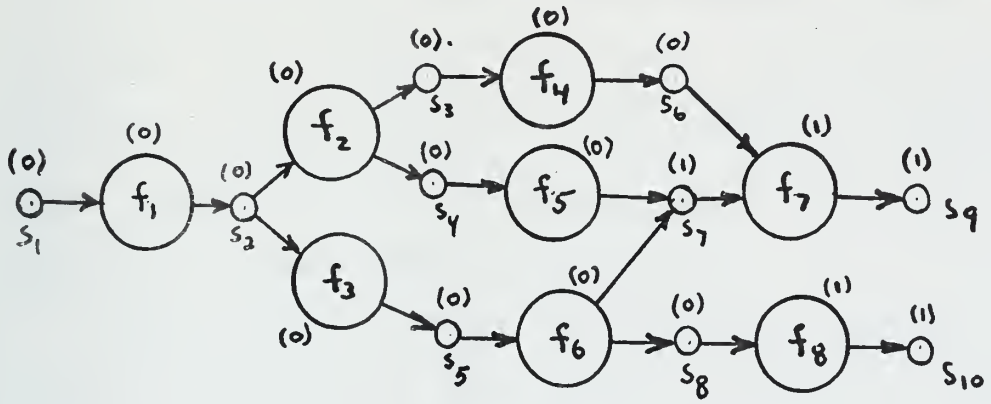

Figure 25a

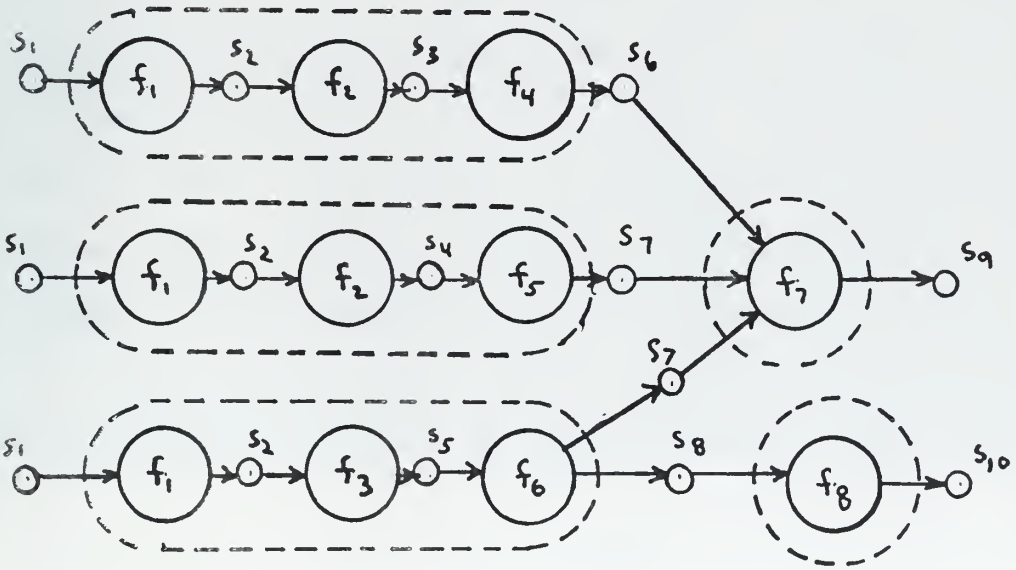

Figure 25b

V. COALESCING THE FUNCTIONAL ELEMENTS WITHIN A LOBE

A. NEW CRITERIA OF CLUSTERING

In the case of a digital circuit without feedback, it is natural to define optimization criteria in terms of delay. The primary objective that seems most natural is the one stated in Sections I and III, namely, minimize the maximum delay caused by external arcs over the longer paths of the circuit.

This is not, in general, the case for the lobes of a circuit with feedback. Maximal paths are difficult to define, since if a cycle is included among the arcs from one vertex to another, an electronic pulse traversing this sequence of arcs might traverse the cycle any number of times. It is certainly true, however, that in the final solution of the partition problem applied to the functional elements of a lobe, some definition of maximal paths will have to be made, and delay across them be considered.

The criteria by which the functional elements of a lobe will be coalesced in this section is stated in terms of a distinguished node of the lobe and the tracks of a lobe. Following Berge [Ref. 1], if u and v are nodes, $u \neq v$, a track from u and v is defined to be a path of minimum length going from u to v. Since a lobe is strongly connected, there always exists a track between any two vertices of a lobe.

The distinguished node of the lobe is, intuitively, that node which represents an output from the lobe, and at which

it is important, according to the logic of the circuit, to have a usable electronic pulse with minimal delay. It may be unrealistic to expect the logic designer to distinguish just one node of a lobe as being the most important if the lobe is very large. Assume, however, that such a distinguished node can be identified.

Considering a lobe in relation to the entire circuit, there might be several arcs which carry signals from functional elements outside the lobe, to functional elements inside the lobe. The criteria of minimizing the delay through the entire circuit is still of prime importance here. In the case when a lobe satisfies the constraints on clustering, and the lobe is contained within a single cluster, then the delay through the entire circuit is not increased by external connections within the lobe. But when the functional elements of the lobe must be placed in more than one cluster, it is of interest to do this in such a way as to minimize the effect of increased delay over the entire circuit. Therefore the distinguished node of the lobe would be that node which lies on an output signal of the lobe, and to which it is most important to minimize delay along a path from any node of the lobe to the distinguished node.

Hence, the criteria by which the functional nodes of a lobe will be clustered is to minimize, for every node in the lobe, the delay between each node and the distinguished node. The algorithm to be produced, which will find a clustering

that satisfies this criteria, does not in any way depend on the physical characteristics of the distinguished node. Hence any node of the lobe could be chosen for this purpose. It would be up to the logic designer, or the computer, to determine the identity of the distinguished node based on the logic of the circuit or some graph theoretic property of the circuit.

B. STATEMENT OF THE ALGORITHM

For the purpose of describing the algorithm, the lobe in Fig. 26 will be used. Suppose that signal node s_4 is the

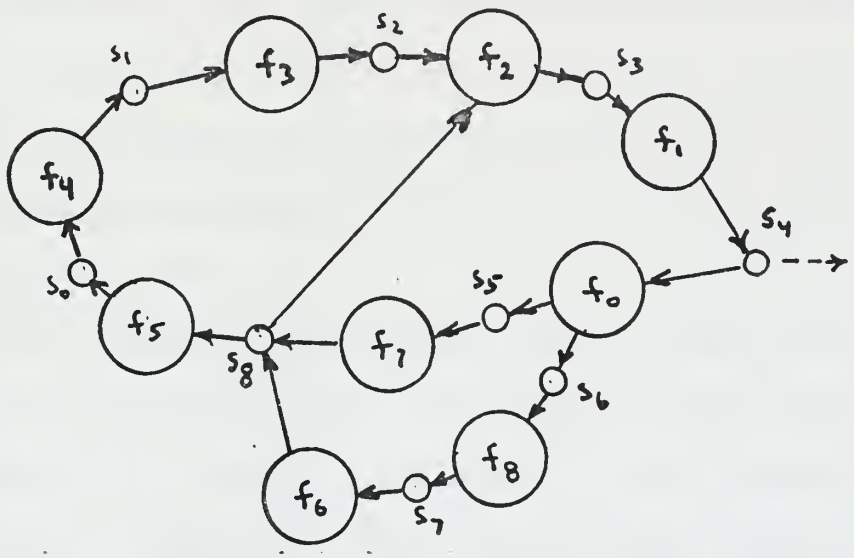

Figure 26

distinguished node. Then it is the goal here to indicate which functional elements are to be coalesced so that the delay along a track (path of minimal length) from any functional node to s_4 is minimized. For illustration it will

be assumed that no more than three functional elements can be within a single cluster.

To determine the cluster, a reverse attainability tree $T_r(s_4)$ is constructed, with the distinguished node s_4 as the root. A reverse attainability tree is similar to an attainability tree, except that instead to placing on the branches those nodes which are attainable from the root, one places on the branches those nodes from which the root is attainable. The reverse attainability tree is built in exactly the same way as was the attainability tree, except that instead of stepping along the paths of the graph from the distinguished node, determining the branch nodes which are reachable by paths of increasing length, one steps along the paths in the opposite direction than indicated by the arcs of the graph. The reverse attainability tree $T_r(s_4)$ for the lobe in Fig. 26 is shown in Fig. 27.

As the reverse attainability tree is being built, if record is kept of the level at which each node first appears in the tree, the reverse attainability tree can be reduced to the tree in Fig. 28. This tree contains each node exactly once, and in each case the appearance of the node is on that branch which represents the shortest path from any node to s_4. If a node appears on different branches at the same level of the reverse attainability tree, then either occurrence can be used in producing the reduced tree. Hence by comparing the tree in Fig. 28 with the graph in Fig. 26, and by following

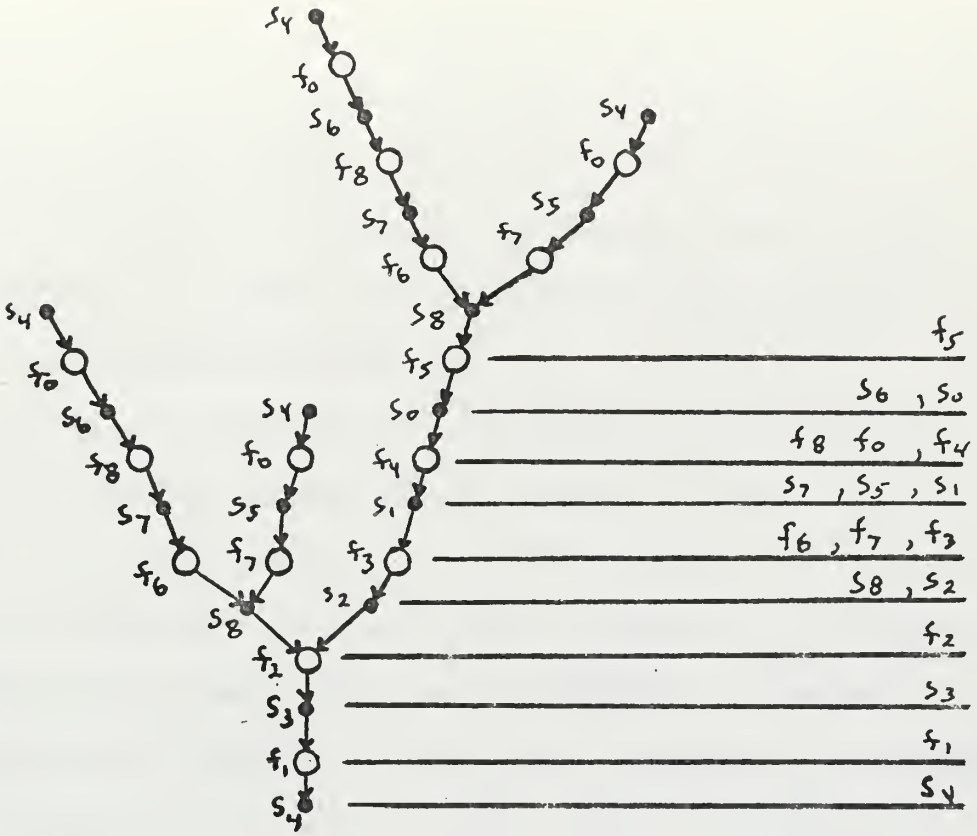

Figure 27

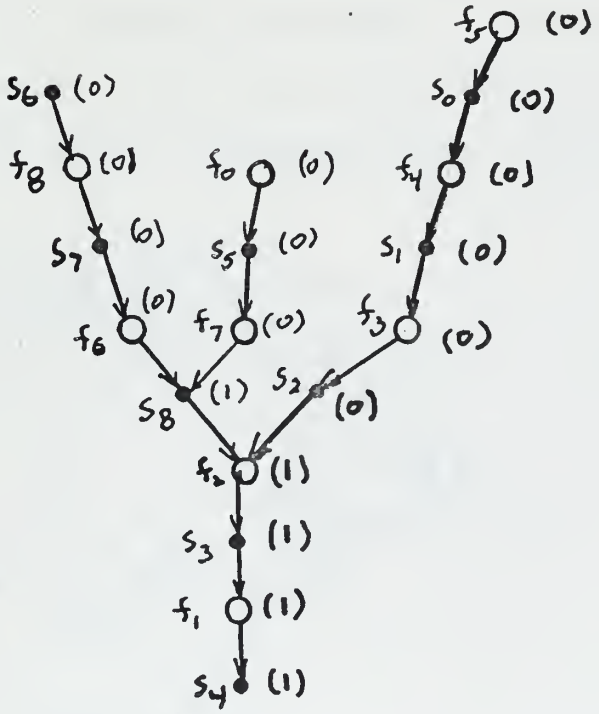

Figure 28

the corresponding paths from any node to s_4, since this path is the path of minimal length from the node to s_4, it is a track in the original graph. Hence the tree in Fig. 28 represents the tracks of Fig. 26. Designate this tree by $T_t(s_4)$, the tree of tracks from each node of the original lobe to the distinguished node s_4.

Once $T_t(s_4)$ is produced, the algorithm is completed by applying the labeling procedure of Section IV A for rooted sink trees. By treating $T_t(s_4)$ as a bi-digraph, the functional nodes are clustered in a way which minimizes the maximum delay through the tree. This has the effect of minimizing the maximum delay along any track in the original bi-digraph. Hence by clustering the nodes in this way the criteria for clustering the functional nodes of the lobes of a bi-digraph has been met.

The result of the labeling procedure is shown by the numbers in parentheses in Fig. 28, and the original bi-digraph is shown in Fig. 29 in its clustered form. In this example

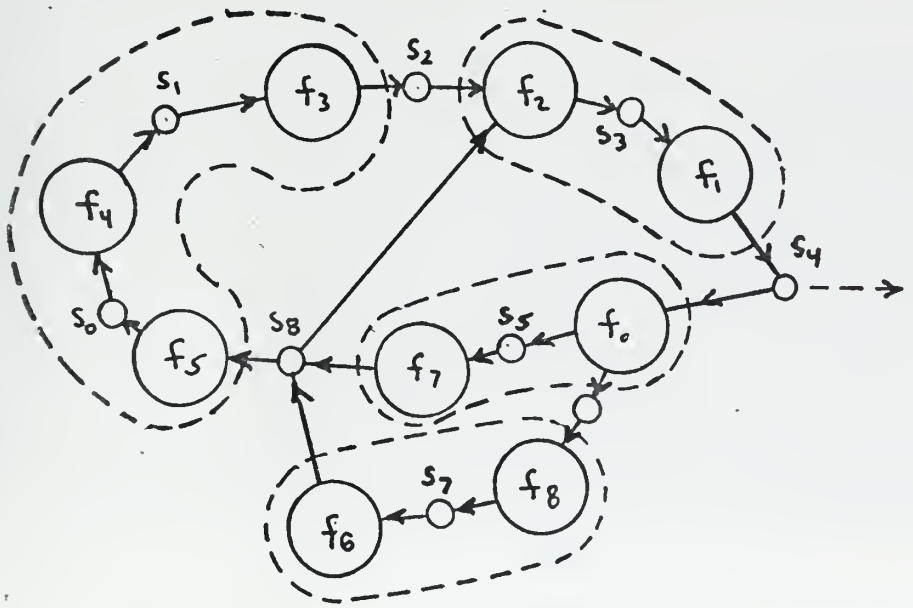

Figure 29

the weights of the functional nodes are assumed to be 1. The maximum number of functional nodes per cluster is three, so M = 3, and any limitations on the number of external connections can be satisfied by simply checking the number of connections to a cluster in the original bi-digraph.

VI. CONCLUDING REMARKS

The solution of the partition problem is far from being complete. In fact, at this point in time, even the statement of the problem is tenuous. Solutions are offered by investigators not yet with the intent of providing the complete answer, but only with the hope of offering a new approach or a new insight, both to refine the definition of the problem, and to advance the solution to the problem as it is now stated.

It is clear that the digital circuits represented in terms of graph theory are the digital circuits of computers yet to be designed, and that the algorithms that are to provide workable solutions to the problems of design must be written with the intent that they will be programmed on the computer. The algorithms that have been presented in this thesis are such that they could be programmed. But much work has yet to be done, and much work is being done, on determining the best methods to represent a graph in the computer, and the best ways to implement algorithms which seem straightforward to humans, but not so to computers.

The computer program which follows is meant to illustrate the results of Section III, the extraction of a lobe from the bi-digraph. It has been included not because it is expected that the data structures which are used would ever be used in a circuit designing system. Rather it has been included because the finding of maximal strongly-connected subgraphs is a problem in itself, which has many other applications besides its part in the partition problem.

Some other topics which are being investigated which closely affect the partition problem include:

i) Finding an adequate solution for general acyclic networks.

ii) Finding alternate optimization criteria for the clustering algorithm.

iii) Determining the best ways to represent a graph in the computer, with emphasis on efficient graph manipulation.

```
C.... THIS PROGRAM WILL DETERMINE THE LORES OF A BI-DIGRAPH.
C.... THE BI-DIGRAPH IS EXPRESSED IN DOUBLE LIST COMPACT
C.... FORM IN THE VECTORS L AND K.  L HAS ONE MORE ENTRY
C.... THAN THE TOTAL NUMBER OF NODES SO THE COMPUTATION OF
C.... THE FINAL ENTRY OF K CAN BE DONE IN THE SAME MANNER
C.... AS THE OTHER ENTRYS.
C....
C....
      IMPLICIT INTEGER (A-Z)
      DIMENSION          L(100),PRED(100),PNEXT(100),
     1SCES(100),SBACK(100),NAME(1800),TREE(2,900),
     2K(1500),STACK(200)
      EQUIVALENCE (TREE(1,1),NAME(1))
C....
C.... READ IN THE NXM BI-DIGRAPH IN DOUBLE LIST COMPACT FORM
C.... THE DI-GRAPH HAS MM ARCS.
      READ(5,1)M,N,MM
 1    FORMAT(3I3)
      H = M + N + 1
      READ (5,2) (L(I),I=1,H)
 2    FORMAT (20I4)
      READ (5,2) (K(I),I=1,MM)
C.... EXTRACT THE DIGRAPH G OF FUNCTIONAL NODES
C.... SET UP LIST STRUCTURES TO SUCCESSIVELY ELIMINATE
C.... SINK AND SOURCE NODES
      DO 10 I = 1,M
      PRED(I) = 0
      PNEXT(I) = 0
      SCES(I) = 0
 10   SBACK(I) = 0
      NPA = 1
      NPB = 1799
      DO 50 I = 1,M
      J = L(I)
      JJ = L(I+1)
 12   IF (JJ-J) 50,50,13
 13   H=K(J)
      HH = H + 1
      H = L(H)
      HH = L(HH)
 14   IF (HH-H) 45,45,15
 15   B = K(H)
      PRED(B) = PRED(B) + 1
      SCES(I) = SCES(I) + 1
      NPA1 = NPA + 1
      NAME(NPA1) = PNEXT(I)
      PNEXT(I) = NPA
      NAME(NPA) = B
      NPB1 = NPB + 1
      NAME(NPB1) = SBACK(B)
      NAME(NPB) = I
      SBACK(B) = NPB
      NPA = NPA + 2
      NPB = NPB - 2
      IF (NPA - NPB) 18,1000,1000
 18   H = H + 1
      GO TO 14
 45   J = J + 1
      GO TO 12
 50   CONTINUE
C.... SUCCESSIVELY REMOVE SINK NODES
      STP1 = 0
      STP2 = 0
      DO 56 I = 1,M
      IF (SCES(I)) 55,55,56
 55   STP1 = STP1 + 1
      STACK(STP1) = I
 56   CONTINUE
 145  STP2 = STP2 + 1
      IF (STP2 - STP1) 150,150,200
```

```
  150   I = STACK(STP2)
        J = SBACK(I)
  155   IF(J) 145,145,160
  160   C = NAME(J)
        SCES(C) = SCES(C) - 1
        IF(SCES(C)) 170,170,188
  170   STP1 = STP1 + 1
        STACK(STP1) = C
  188   J = J + 1
        J = NAME(J)
        GO TO 155
C.... SUCCESSIVELY REMOVE ALL SOURCE NODES
  200   STP3 = STP1
        STP2=STP1
        DO 240 I = 1,M
        IF (PRED(I)) 240,235,240
  235   STP1 = STP1 + 1
        STACK(STP1) = I
  240   CONTINUE
  245   STP2 = STP2 + 1
        IF(STP2 - STP1) 260,260,300
  260   I = STACK(STP2)
  263   J = PNEXT(I)
  265   IF (J) 245,245,270
  270   C = NAME(J)
        IF (SCES(C)) 275,275,280
  275   NAME(J) = -1
        GO TO 285
  280   PRED(C) = PRED(C) - 1
         IF (PRED(C)) 290,290,285
  290   STP1 = STP1 + 1
        STACK(STP1) = C
  285   J = J + 1
        J = NAME(J)
         GO TO 265
  300   IF(STP3) 330,330,310
  310   DO 320 I = 1,STP3
        J = STACK(I)
  320   PRED(J) = 0
  330   CONTINUE
C.... BUILD THE ATTAINABILITY TREES AND
C.... DETERMINE THE GRAPH'S SIMPLE CYCLES
        L1 = M + 1
        DO 400 I = 1,L1
  400   STACK(I) = 0
        T1 = NPA1/2
        L1 = NPA1 + 1
        DO 401 I = L1,1800
  401   NAME(I) = 0
        STP1 = 0
C.... PICK A ROOT NODE
  410   STP1 = STP1 + 1
        IF(STP1 - M) 412,412,600
  412   IF (STACK(STP1)) 1010,415,410
  415   IF (PRED(STP1)) 410,410,420
  420   I = T1
        J = I + 1
        Q = 200
        NPA = 1800
C.... BUILD THE TREE USING T(1,I) TO CONTAIN A VERTEX,
C.... T(2,I) A POINTER
        TREE(1,I+1) = STP1
  495   I = I + 1
        IF (I-J) 500,500,410
  500   IF (TREE(2,I)) 495,501,501
  501   C = PNEXT(TREE(1,I))
  505   IF (C) 495,495,510
  510   CC = NAME(C)
        IF (CC) 540,511,511
  511   J = J + 1
        TREE(1,J) = CC
        TREE(2,J) = I
```

```
      STACK(CC) = 1
C.... CHECK FOR A SIMPLE CYCLE
      II = J
  515 II = TREE(2,II)
      IF (CC - TREE(1,II)) 525,520,525
  520 STACK(Q) = NPA
      Q = Q - 1
      II = J
      TREE(2,II) = -TREE(2,II)
  521 II = IABS(TREE(2,II))
      NAME(NPA) = TREE(1,II)
      NPA = NPA - 1
      IF(CC - TREE(1,II)) 521,540,521
  525 IF (TREE(2,II)) 515,540,515
  540 C = C + 1
      C = NAME(C)
      GO TO 505
600   CONTINUE
C.... NOW FORM THE LOBES
      I = 0
  795 Q = Q + 1
      IF(Q-200) 796,910,910
  796 IF(NAME(NPA+1)+1) 799,798,798
  799 NPA = STACK(Q)
      GO TO 795
  798 NPA = NPA + 1
  800 I = I + 1
      STACK(I) = NAME(NPA)
      IF(NPA-STACK(Q)) 798,798,805
  805 IF(Q-200) 810,900,900
  810 NPB = NPA
      NPA = NPA - 1
      P = Q + 1
      NPB = NPB + 1
      L1 = STACK(P)
  815 DO 840 II = NPB,L1
      DO 835 J = 1,I
      IF(NAME(NPB) - STACK(I)) 835,880,835
  835 CONTINUE
  840 CONTINUE
      GO TO 890
  880 I = I + 1
      STACK(I) = NAME(NPB)
      NAME(NPB) = -1
      NPB = NPB + 1
      IF (NPB-STACK(P)) 880,880,890
  890 NPB = STACK(P) + 1
      P = P + 1
      IF (P-200) 815,900,900
  900 WRITE(6,901) (STACK(L1),L1=1,I)
  901 FORMAT (' THIS IS A LOBE',/,20I4)
      GO TO 795
  910 STOP
 1000 WRITE(6,1001)
 1001 FORMAT(' OVERFLOW ERROR')
      STOP
 1010 WRITE (6,1011)
 1011 FORMAT(' STACK ERROR')
      STOP
      END
```

BIBLIOGRAPHY

1. Berge, C., The Theory of Graphs and Its Applications, John Wiley and Sons, 1962.

2. Breuer, M. A., "General Survey of Design Automation of Digital Computers," IEEE Proceedings, v. 54, pp. 1708-1721, December 1966.

3. Busacker, R. G. and Saaty, T. L., Finite Graphs and Networks: An Introduction with Applications, McGraw-Hill, 1965.

4. Caldwell, S. H., Switching Circuits and Logical Design, John Wiley and Sons, 1958.

5. Cochrane, J., On the Determination of Cycles in Digraphs, M.S. Thesis, Naval Postgraduate School, Monterey, 1970.

6. DiGiulio, H. A. and Tuan, P. L., "A System for Network Picture Processing with Interactive Computer Graphics," Proceedings of 24th National Conference Association for Computing Machinery, pp. 597-609, 1969.

7. Harary, F., Graph Theory and Theoretical Physics, Academic Press, 1967.

8. Harary, F. and Ross, I. C., "A Procedure for Clique Detection Using the Group Matrix," Sociometry, v. 20, pp. 205-215, 1957.

9. Kamae, T., "Notes on a Minimum Feedback Arc Set," IEEE Transactions on Circuit Theory, v. CT-14, No. 1, p. 78-79, March 1967.

10. Kodres, Uno R., "Logic Circuit Layout," The Digest Record of the 1969 Joint Conference on Mathematical and Computer Aids to Design, IEEE No. 69063-C, pp. 165-191, Oct. 1969.

11. Lampel, A. and Cederbaum, I., "Minimum Feedback Arc and Vertex Sets of a Directed Graph," IEEE Transactions on Circuit Theory, v. CT-13, pp. 399-403, December 1966.

12. Lawler, E. L., Levitt, K. N., and Turner, J., "Module Clustering to Minimize Delay in Digital Networks," IEEE Transactions on Computers, v. C-18, No. 1, pp. 47-57, January 1969.

13. Luccio, F. and Mariagiouanna, S., "On the Decomposition of Networks in Minimally Interconnected Subnetworks," IEEE Transactions on Circuit Theory, v. CT-16, No. 2, May 1969.

14. Ramamoorthy, C. V., "Analysis of Graphs by Connectivity Considerations," Journal of the Association for Computing Machinery, v. 13, No. 2, pp. 211-222, April 1966.

15. Seshu, S. and Reed, M., Linear Graphs and Electrical Networks, Addison-Wesley, 1961.

16. Welch, J. T., "A Mechanical Analysis of the Cyclic Structure of Undirected Graphs," Journal of the Association for Computing Machinery, v. 13, No. 2, pp. 205-210, April 1966.

17. Younger, D. H., "Minimum Feedback Arc Sets for a Directed Graph," IEEE Transactions on Circuit Theory, v. CT-10, No. 2, pp. 238-245, June 1963.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22341	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Assoc. Professor U. R. Kodres, Code 53 Kr Department of Mathematics Naval Postgraduate School Monterey, California 93940	1
4.	LTJG Thomas J. Breckon, USN 4300 Soquel Drive Soquel, California 95073	1

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1 ORIGINATING ACTIVITY *(Corporate author)*

Naval Postgraduate School
Monterey, California 93940

2a. REPORT SECURITY CLASSIFICATION

Unclassified

2b. GROUP

3 REPORT TITLE

DETERMINING PARTITION ELEMENTS WITH FEEDBACK CONSTRAINTS

4 DESCRIPTIVE NOTES *(Type of report and, inclusive dates)*

Master's Thesis; June 1970.

5. AUTHOR(S) *(First name, middle initial, last name)*

Thomas Joseph Breckon
Lieutenant (junior grade), United States Navy

6. REPORT DATE

June 1970

7a. TOTAL NO. OF PAGES

76

7b. NO. OF REFS

17

8a. CONTRACT OR GRANT NO.

b. PROJECT NO.

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES

12. SPONSORING MILITARY ACTIVITY

Naval Postgraduate School
Monterey, California 93940

13. ABSTRACT

The partition problem is that step in the layout problem in which it must be decided which of the elementary digital circuits are to be coalesced into a single, electronic package. A solution of the partition problem must satisfy constraints on the maximum number of elementary circuits that can be put into a single package, and on the number of external connections that can be attached to the package.

A solution due to Lawler et al [Ref. 12], which minimizes delay caused by clustering electronic elements, is extended to cyclic networks. A new algorithm to extract from a graph the maximal strongly-connected subgraphs (lobes) is developed, and a new approach to clustering the digital elements of a lobe is presented. The digital circuit is represented by a bipartite graph, and solutions are expressed in terms of graph theory.

KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Computer						
Design						
Graph Theory						
Layout						
Partition						
Feedback						
Clustering						

Thesis
B80323 Breckon
c.1
119297
Determining partition elements with feedback constraints.